ABIMAEL ALVES DE OLIVEIRA JUNIOR

VISUALIZADOR 3D PARA DADOS DE RADAR METEOROLÓGICO USANDO WEBGL

CURITIBA

Outubro/2012

ABIMAEL ALVES DE OLIVEIRA JUNIOR

VISUALIZADOR 3D PARA DADOS DE RADAR METEOROLÓGICO USANDO WEBGL

Dissertação apresentada ao Programa de Pós-Graduação em Métodos Numéricos em Engenharia (PPGMNE) da Universidade Federal do Paraná (UFPR), como parte dos requisitos para obtenção do título de Mestre em Ciências na área de concentração Mecânica Computacional

Orientador: Prof. Dr. Sérgio Scheer

CURITIBA

Outubro/2012

Termo de Aprovação

ABIMAEL ALVES DE OLIVEIRA JUNIOR

VISUALIZADOR 3D PARA DADOS DE RADAR METEOROLÓGICO USANDO WEBGL

Dissertação aprovada como requisito parcial para obtenção do título de Mestre em Ciências, na área de concentração Mecânica Computacional, do Programa de Pós-Graduação em Métodos Numéricos em Engenharia (PPGMNE) da Universidade Federal do Paraná (UFPR), pela comissão formada pelos professores:

Prof. Dr. Sérgio Scheer
Programa de Pós-Graduação em Métodos Numéricos em Engenharia - Universidade Federal do Paraná

Prof. Dr. Klaus de Geus
Programa de Pós-Graduação em Métodos Numéricos em Engenharia - Universidade Federal do Paraná

Prof. Dr. Walmor Cardoso Godoi
Programa de Pós-Graduação em Desenvolvimento de Tecnologia IEP-Lactec

Dedicatória

Dedico este trabalho primeiramente a Deus, a quem rendo louvor e adoração,
a minha esposa, Léa, e meu filho, Abimael Neto, que estiveram comigo em todos os momentos nesta jornada, sonhando e torcendo. Este título também é de vocês!
Também dedico a meus pais, que me proveram o bem mais importante: a vida.

Agradecimentos

Este período de estudos para obtenção deste título só foi possível porque contei com o apoio e ajuda de várias pessoas.

Agradeço primeiro a Deus, que me permitiu sonhar e abençou-me de tal maneira que o sonho hoje torna-se realidade.

Agradeço à minha esposa, Léa, que desde os primeiros momentos, acompanhou e torceu muito. Depois, durante a jornada, compreendeu e ajudou-me nas privações de ordem financeira e de ordem temporal. A você, meu amor, meu agradecimento!

Agradeço a meu filho, Abimael Neto, que também acompanhou as privações. A você meu filho, meu muito obrigado!

Agradeço ao meu orientador, professor doutor Sergio Scheer, pelas suas orientações, conselhos e auxílios em momentos que foram importantíssimos. Muito obrigado!

Agradeço ao senhor Fábio Sato do Simepar pelas oportunidades de aprendizado.

Agradeço aos senhores César Benneti, Leornardo Calvetti e Reinaldo Silveira do Simepar pelos comentários e sugestões.

Agradeço a Capes pelo suporte financeiro.

Agradeço ao Simepar pelo suporte financeiro e pelos dados que permitiram que este trabalho fosse realizado.

Epígrafe

Como é feliz o homem que busca sabedoria, o homem que obtém entendimento
Provérbios 3.13
Ensina-nos a contar os nossos dias
para que nosso coração alcance sabedoria
Salmos 90.12

SUMÁRIO

Lista de Figuras

Lista de Siglas

API	API : Application Programming Interface - Interface para Programação de Aplicações. Conjunto de especificações para permitir comunicação entre aplicações ou componentes de *software*
HTML	HyperText Markup Language ou Linguagem de Marcação de Hipertexto. Linguagem de marcação para criação de páginas da Internet
CAD	Computer Aided Design: Projeto auxiliado por computador. Programa ou sistema computacional que permite a modelagem, projeto e desenvolvimento de objetos, desenho técnicos para Engenharia e Design utilizando o computador como ferramenta de desenho.
VRML	Virtual Reality Modeling Language, linguagem de marcação para realidade virtual. Padrão desenvolvido em 1994 para definição de objetos 3D a partir de texto com sintaxe específica que permitia definir os objetos e modelá-los para criação de ambientes de realidade virtual (RAGGETT, 1994).
ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
XML	XML é a sigla obtida a partir dos termos em inglês eXtended Markup Language : notação baseada em rótulos e definições. Utilizada para troca de mensagens entre aplicações.

Resumo

A previsão de tempo tem exercido um importante papel na prevenção de perdas de vidas e de bens materiais devido a alagamentos, inundações ou desmoronamentos causados por eventos severos tais como chuvas intensas ou tempestades. Porém as ferramentas de visualização de dados de Radar Meteorológico são bidimensionais, ou seja, apresentam os dados de forma planar. Além disto, as ferramentas tem necessidade de instalação e muitas vezes, são desenvolvidas para Sistemas Operacionais específicos. Este trabalho apresenta uma ferramenta de Visualização de Dados de Radar Meteorológico Tridimensionais utilizando WebGL. A ferramenta apresentada permite visualizar os dados atualizados de leitura da atmosfera da região do Radar Meteorológico, em uma Visualização Tridimensional, permitindo visualizar as estruturas das nuvens e chuva, por meio da reflectividade captada pelo Radar. Utilizando WebGL, a visualização pode ser feita sem necessidade de instalação de nenhum tipo de software ou plugin especial, bastando um navegador de Internet compatível com WebGL, tal como Google Chrome© e Mozilla Firefox©. Mesmo sem possuir o conceito de câmera em seu conjunto de instruções, WebGL permite a programação com desenvolvimento de funções em JavaScript para que seja possível a implementação de mecanismos de navegação pelos dados visualizados. Com este recurso, o profissional pode navegar dentro do volume de dados, permitindo visualização e análise mais detalhada de algum evento metereológico que deseja-se averiguar usando os controles da câmera virtual. Os experimentos constituiram-se de testes visualizando treze elevações de data e hora específicas separadamente, visualizando apenas uma elevação (sweep) em cada teste. Foi efetuado também, experimento com todas as treze elevações para a mesma data e hora específicas, com todos os dados e formando o volume tridimensional. O visualizador teve bom desempenho, inclusive com a navegação pela câmera. A ferramenta de filtro foi utilizada para permitir a remoção de parte dos dados, disponibilizando melhor visualização de dados específicos. Baseado nos depoimentos coletados, a ferramenta tem potencial para uso tanto em ambiente operacional quanto em ambiente de pesquisa, e a ferramenta de navegação com a câmera, permite a visualização de detalhes não facilmente identificados com as ferramentas de visualização atualmente utilizadas. Conclue-se que a ferramenta apresentada tem bom desempenho para ser empregada para previsões de curto prazo (Nowcast) e também em ambiente de pesquisa, permitindo visualização tridimensional, compreensão e análise dos fenômenos metereológicos, utilizando tecnologia preparada para a Internet, sem instalação de *software*, pacote ou *plugins*.

Palavras-chave: Visualização Científica; 3D; WebGL; Radar Meteorológico, Previsão, Nowcast .

Abstract

Weather forecast has been playing an important role preventing loss of life and goods due calamity and destruction caused by flood and storms. However, the tools for data visualization of Weather Radar are bidimensional, that means, they provide visualization only in a planar way. Besides, these tools need installation and several times, they are developed for specific Operational Systems. This work presents Three-dimensional Viewer of Weather Radar Data using WebGL. The tool presented allows visualize up-to-date data collected from atmosphere by the Weather Radar in a tridimensional view, allowing view the structures of clouds and rains, by the Reflectivity measured by the Weather Radar. Using WebGL, the view can be done without any installation of programs or plugins, only a Internet browser compatible as is Google Chrome© or Mozilla Firefox©. Despite the fact that WebGL does not have a camera, it allows the development of functions in JavaScript that make possible implementation of mechanisms for navigation inside the data renderized. In addition, the profissional can navigate inside the data volume allowing a better analysis with more details for some weather event that one wants to verify using the camera controls. The experiments done were tests with thirteen sweeps for a specific date and time separately, displaying only one sweep per time in each test. Also, was done other experiment with same data and time with all data of the thirteen sweeps forming 3D Volume. The viewer had good performance, including camera movements. The filter tool was used to remove some data, allowing visualization for specific data. Based in the testimonies collected, the viewer has good potential to be used both in Operational Environment as well in an Research Environment. As conclusion, the tool proposed in this work has good performance for the operational environment for short-range forecast (Nowcast) and for the research environment allowing 3D Visualization, understanding and analysis of weather phenomena, using tecnology ready to the Internet, without installation of software, package or plugins.

Scientific Visualization; WebGL; Weather Radar; 3D Viewer; Nowcast.

1 INTRODUÇÃO

A maior parte do território brasileiro encontra-se situado entre o Trópico de Capricórnio e o Equador, e portanto com clima Tropical e Equatorial. Com esta posição geográfica, o país caracteriza-se por ter clima com períodos de intensas chuvas, principalmente durante o verão. Muitas vezes, as chuvas trazem alagamentos, inundações e destruição, principalmente nas camadas da sociedade menos favorecidas.

De janeiro a março de 2011, diversos estados brasileiros sofreram alagamentos, em particular a região serrana do Estado do Rio de Janeiro, onde houveram mais de novecentas mortes (FOLHA DE SÃO PAULO, 2011) e também o litoral do Paraná (PRATEANO; RUPP; GARMATTER, 2011), em particular as cidades de Antonina e Paranaguá. Tragédias como essas não são exclusividade do Brasil. Em janeiro de 2011, a Austrália sofreu graves enchentes (FOLHAPRESS, 2011), aplacando a vida de milhares de pessoas e causando prejuízos materiais de grande monta.

Esses fenômenos meteorológicos, muitas vezes comuns na primavera e verão na Região Sul, são alterados e influenciados por fenômenos como El Niño e La Niña (NERY, 2005). No caso do fenômeno El Niño, há um aumento nas precipitações e, por consequência, incremento na probabilidade de enchentes.

Além da intensidade dos fenômenos atmosféricos terem uma variação, seus efeitos são potencializados pela ação do homem. A impermeabilização do solo e ocupação das margens dos rios tem incrementado os efeitos de enchentes e impactos na sociedade das grandes cidades, afetando não apenas suas camadas mais inferiores, mas também as chamadas classes média e alta (CASTELLANO; NUNES, 2010).

Os sistemas de previsão de tempo são bem compreendidos e permitem, tanto por modelos numéricos, quanto por visualização de imagens de satélite e dados de estações, que os meteorologistas possam prever as condições climáticas para os próximos dias, porém para intervalos de tempo da ordem de dias. Porém, a visualização de dados do Radar Meteorológico permite a previsão dos eventos que estão ocorrendo em uma área de abrangência menor que as informações fornecidas pelas imagens de satélite e modelos numéricos e em intervalo de tempo da ordem de poucas horas. Esta forma de previsão permite construir sistemas de alertas para as

populações que possam ser atingidas por algum evento meteorológico severo, visto que pode-se monitorar o evento em intervalos de tempo menores. Contudo, estes sistemas denominados de Nowcast precisam ser aprimorados, por meio da melhoria das ferramentas que o meteorologista tem em mãos, para que este possa efetuar previsões acuradas e, se for necessário, disparar alertas para que as autoridades tomem ações necessárias. Atualmente, temos uma melhoria nos métodos numéricos para previsão mediante modelos, contudo o futuro nos aponta para melhoria de ferramentas voltadas para o meteorologista (WILSON, 2004) de modo que permita observar e perceber melhor o desenvolvimento dos fenômenos, visando avisar ou alertar as populações, prevenindo perdas materiais e de vidas.

Tem-se que as ferramentas disponíveis para uso em ambiente operacional estão sempre vinculadas a programas que dependem de suporte do sistema operacional do usuário e não têm disponibilidade de visualização por meio da Internet. Este estudo busca justamente desenvolver uma ferramenta que permita que os dados do radar possam ser lidos remotamente e visualizados por meio da Internet em ambiente tridimensional, sem as limitações para instalação de programas ou bibliotecas que trazem complexidade e muitas vezes incompatibilidades.

1.1 OBJETIVOS

Percebe-se então, que é necessário o desenvolvimento de técnicas e do uso de equipamentos que possam auxiliar agências governamentais e de pesquisa no monitoramento do tempo para aprimorar a previsão de eventos considerados severos.

O objetivo geral do trabalho é desenvolver uma ferramenta de Visualização de Dados 3D utilizando WebGL.

Como objetivos específicos, tem-se:

- Desenvolver protótipo de ferramenta de Visualização de Dados de Radar Meteorológico 3D, porém com capacidade para apresentação dos dados em tempo hábil de modo que possa ser utilizado em ambiente operacional de análise de eventos meteorológicos;
- Permitir a visualização por meio da Internet e dentro de navegador (*browser*), sem nenhuma instalação de *software* adicional;
- O trabalho limita-se apenas a desenvolver um protótipo, voltado para visualização de dados de refletividade Z de radar meteorológico.

1.2 ESTRUTURA

O trabalho está dividido em 6 capítulos, incluindo este capítulo de Introdução.

O Capítulo 1 apresentou a introdução ao tema, bem como os objetivos do trabalho.

O Capítulo 2 apresenta o radar meteorológico, seus princípios físicos de funcionamento e a grandeza refletividade Z, que é lida pelo radar.

O Capítulo 3 apresenta a linguagem WebGL que é a linguagem utilizada para desenvolvimento do sistema para o ambiente Web.

O Capítulo 4 apresenta o desenvolvimento do visualizador, obtenção e preparação dos dados, bem como o exemplo de visualização.

O Capítulo 5 apresenta os resultados obtidos e depoimentos coletados durante fase de testes.

O Capítulo 6 apresenta as conclusões e propostas para trabalhos futuros.

2 RADAR METEOROLÓGICO

Radar é um acrônimo em inglês para *Radio Detection and Ranging*, que pode ser traduzido como "Detecção e estimativa de distância por rádio". Tem suas origens nas pesquisas com rádio. Taylor e Young, dois engenheiros que trabalhavam com rádio para a Marinha Americana no ano de 1922, perceberam que quando navios passavam entre o transmissor e o receptor, havia reflexões do sinal. Em 1930, Taylor escreveu um relatório para a Marinha sobre eco de sinais de rádio de objetos em movimento, que conduziu ao desenvolvimento do radar (IEEE, 2012).

O radar teve grande desenvolvimento durante a 2ª Guerra Mundial, sendo considerado por alguns autores como "a invenção que mudou o mundo" (BUNDERI, 1997), apesar de ser uma invenção ainda em desenvolvimento (KRAUSE, 2000).

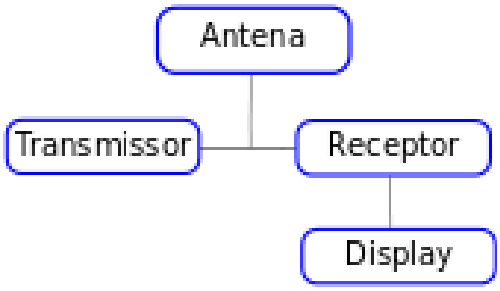
Com o final da 2ª Guerra Mundial, muitos dos radares residuais e descartados pelos militares puderam ser adquiridos para uso civil e os interessados em fazer pesquisas em Meteorologia utilizando o radar foram os primeiros a adquiri-los (RINEHART, 2004, p. 3). Desde então, o radar para uso em Meteorologia vem evoluindo de forma constante, primeiramente na parte de seus equipamentos e recentemente em aplicativos de *softwares* especializados (RINEHART, 2004, p. 3).

2.1 FUNCIONAMENTO DO RADAR

Como bem estabelecido por Taylor, o radar funciona basicamente pela detecção do eco dos sinais eletromagnéticos de objetos. Também pode ser entendido pela analogia com o sistema de orientação utilizado pelos morcegos, que emitem sons em alta frequência e captam seu eco , identificando assim do obstáculo (SILVA NETO, 2008).
A figura 1 contém um diagrama em blocos de um radar simplificado.

O Transmissor é responsável pela geração e envio do pulso (ou *burst*) de ondas eletromagnéticas. Os três tipos mais importantes de transmissores são: Magnetrom, Klystrom e

Figura 1: DIAGRAMA EM BLOCOS BÁSICO DO RADAR.

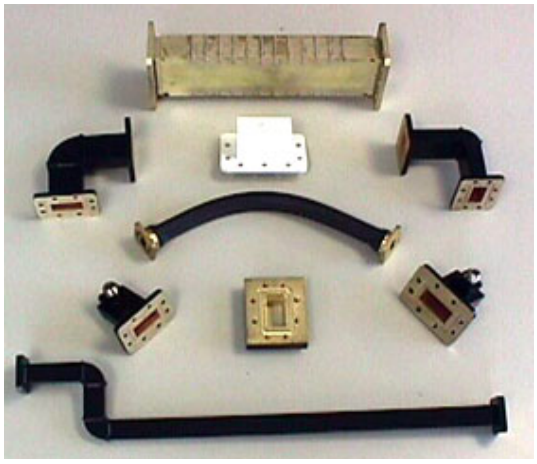


FONTE: o autor

Transmissores de estado sólido. Os dois primeiros tipos foram desenvolvidos a partir de 1939 (RINEHART, 2004, p.18), produzindo sinais eletromagnéticos de alta potência. Recentemente, os transmissores de Estado Sólido produzem baixa potência, mas podem ser conectados em forma de *array* e, com o controle sobre cada elemento, permitem produzir o efeito desejado com altas potências.

As ondas eletromagnéticas são conduzidas até a antena por meio do guia de onda. Diferentemente do que se pode imaginar, devido à alta-frequência utilizada, fios condutores tornam-se inviáveis para conduzir o sinal até a antena. É necessário o uso de um dispositivo similar a um tubo, composto por um material metálico em forma de tubo para conduzir o sinal do transmissor até a antena. A figura 2 mostra alguns componentes para formação do guia de onda. A antena então irradia o sinal para a atmosfera.

Figura 2: ELEMENTOS DE GUIAS DE ONDA



FONTE: www.antek.com

A antena de radar normalmente é identificada pelo conjunto antena-refletor. O dispositivo antena é pequeno quando comparado com o tamanho do refletor e é responsável por emitir as ondas eletromagnéticas e captar as ondas refletidas pelos alvos. Uma antena que emitisse as ondas em todas as direções não seria muito prática para captação dos ecos. O melhor é que seja direcional, ou seja, emita e capte ondas eletromagnéticas para e de posição específica. Para isto, a antena é montada em conjunto com o refletor (RINEHART, 2004, p.27).
Há diversos tipos de refletores sendo o mais comum o refletor parabólico. Este é montado de modo que a antena fique no foco do parabolóide. O objetivo desta montagem é que as ondas emitidas pela antena sejam refletidas pelo refletor e todos as ondas que chegam até o refletor sejam refletidas para o foco, onde está posicionada a antena.

As ondas eletromagnéticas transmitidas pela antena viajam até "chocar-se" com algum obstáculo. Dependendo da estrutura física (tipo de material, densidade, etc), parte delas é absorvida e outra parte é refletida. Esta parte das ondas que é refletida é denominada "eco" e são espalhadas em várias direções.

Uma parte da onda refletida é captada pela antena e é enviada ao Receptor que, por meio de um processador de sinais, decodifica e armazena em forma de arquivos ((RINEHART, 2004, p.13).

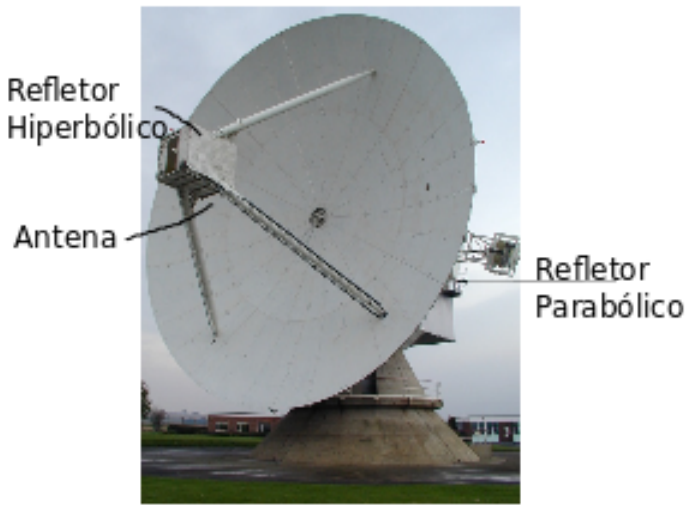
O módulo denominado *Display* é responsável por apresentar a informação captada ao usuário. É uma combinação de hardware e software, onde o arquivo contendo a leitura dos sinais captados pela antena são apresentados ao usuário. A seção 2.2 descreve uma forma de apresentação dos dados de radar no display do usuário.

2.1.1 Princípios Físicos

O funcionamento do radar é baseado nas ondas eletromagnéticas que são capturadas pela antena após refletirem em objetos. Isso é possível graças a um conjunto de fenômenos físicos tais como Reflexão das ondas e Espalhamento.

Da dinâmica ondulatória, sabe-se que ondas podem sofrer reflexão e refração quando a onda alcança uma superfície que separa dois meios. Dependendo das características de cada um dos meios, parte da onda que atinge esta superfície (denominada onda incidente) é refletida (ALONSO; FINN, 1972, p.347). A figura 4 mostra ondas incidentes e ondas refletidas. Isso ocorre porque o meio onde as ondas estão incidindo oferece uma "resistência" ao movimento ondulatório ou a transferência de energia da onda.

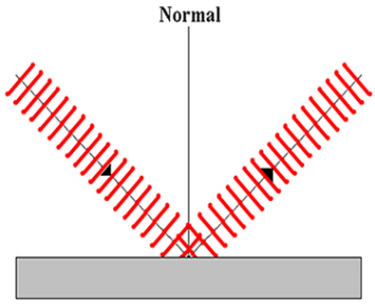
Figura 3: ANTENA DO RADAR DO OBSERVATÓRIO CHILBOLTON



FONTE: Wikipedia

Essa característica do comportamentos das ondas é facilmente notada em ondas mecânicas (ou que necessitem de um meio físico para propagação), como é o caso de ondas em um lago causadas pelo impacto de uma pedra em sua superfície: quando as ondas chegam até a margem, ela oferece a "resistência" ao movimento e portanto, as ondas voltam, gerando outras ondas. O ângulo de reflexão é igual ao ângulo de incidência, espelhado em relação à normal à superfície (ALONSO; FINN, 1972, p.348).

Figura 4: ONDAS INCIDENTES E ONDAS REFLETIDAS



Espalhamento é o fenômeno pelo qual um objeto que é atingido pelas ondas eletromagnéticas sofre perturbação devido aos campos elétrico e magnético da onda e, por conta dessa perturbação, o objeto passa a irradiar ondas eletromagnéticas em todas as direções, podendo as ondas estar ou não com a mesma polarização das ondas incidentes.

Basicamente, tudo o que é visto é fruto de espalhamento da luz. Com exceção de olhar diretamente para o sol, ou para um filamento incandescente de uma lâmpada bem limpa, o

resto que é visto deve-se ao espalhamento da luz pela superfície dos materiais, sejam líquidos ou sólidos. Até o azul que é observado no céu é fruto de espalhamento (Espalhamento de Rayleigh). Esse fenômeno é observado em todos os comprimentos de onda das ondas eletromagnéticas, de cujo espectro, a luz visível é apenas uma parte (MCCARTNEY, 1976, p. 2).

O espalhamento é devido aos conjuntos de átomos e moléculas que constituem os materiais. No caso da atmosfera, ele é devido às moléculas dos vários gases, materiais particulados e aerosóis que estão presentes, bem como gotas de água, gelo, neve e outros elementos de vários tamanhos.

Por definição, a Área de Espalhamento Diferencial é a razão do fluxo da intensidade de energia da onda espalhada por ângulo sólido pelo fluxo incidente (NEWTON, 2002, p. 15) :

$$\frac{d\sigma}{d\Omega} = \frac{I_{espalhado}}{I_{incidente}} \tag{2.1}$$

Mie [1] desenvolveu em 1908 a Teoria de Espalhamento para Esfera, que leva seu nome, a partir da solução das Equações de Maxwell [2], que descreve os campos eletromagnéticos internos e espalhados a partir da iluminação de uma esfera (KNOX, 2011, p. 69) denominada alvo. Por sua teoria, a Área Transversal de Espalhamento (*Scattering Cross Section*) ou Retroespalhamento σ_s do alvo é determinada, entre outros fatores físicos relacionados com o campo elétrico e magnético, pela razão entre o tamanho da esfera e comprimento de onda. Esta razão é dada pela equação 2.2: (NEWTON, 2002, p.49)

$$x = \frac{2\pi R}{\lambda} \tag{2.2}$$

onde R é o raio da esfera e λ é o comprimento da onda eletromagnética que atinge o alvo.

A teoria de Mie pode então, ser utilizada para identificar o espalhamento σ_s em função de seu raio e também do comprimento de onda (KNOX, 2011, p.70). Dependendo do valor da relação $\frac{2R}{\lambda}$, diferentes expressões para σ_s são determinadas. A figura 5 apresenta um gráfico que contém as "regiões" determinadas pela relação $R << \lambda$.

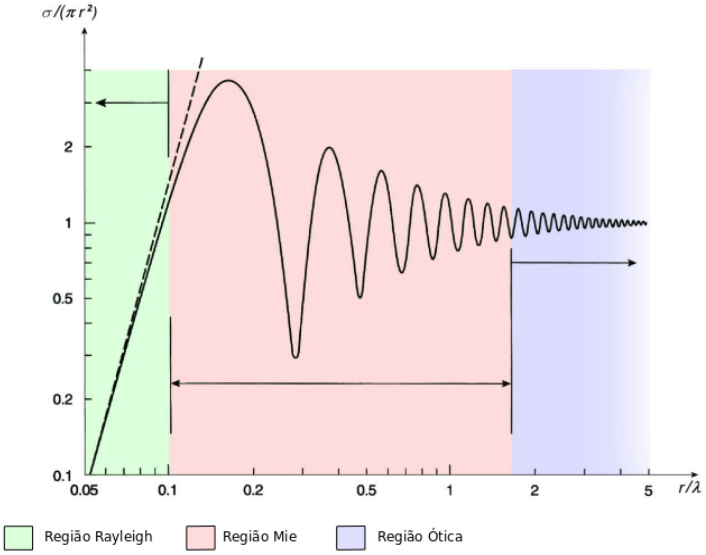
Quando as esferas são pequenas e a relação $\frac{2R}{\lambda} < 0,1$ (RINEHART, 2004, p.73) implica que $x << 1$ e o espalhamento é denominado Espalhamento de Rayleigh em homenagem a Lord Rayleigh[3] que demonstrou esse espalhamento e demonstrou que a expressão σ_s é proporcional a $(2R)^6$.

[1] Gustav Adolf Feodor Wilhelm Ludwig Mie (1869 - 1957); Físico alemão.

[2] James Clerk Maxwell (1831-1879); Físico e matemático escocês conhecido principalmente pelas suas quatro famosas Equações para o Eletromagnetismo.

[3] John William Strutt, Lord Rayleigh (1842-1919) publicou em 1871 [Philosophical Magazine 41(4), p. 107] a explicação que a luz espalhada pelos gases é inversamente proporcional a λ^{-4}.

Figura 5: ESPALHAMENTO σ EM FUNÇÃO DA RELAÇÃO $\frac{R}{\lambda}$



FONTE: www.radartutorial.eu

A expressão completa é dada pela equação: (RINEHART, 2004, p.73)

$$\sigma_s = \frac{\pi^5 |K|^2 (2R)^6}{\lambda^4} \tag{2.3}$$

onde $|K|^2$ é um parâmetro relacionado com o índice complexo de refração do material ou meio.

Para a Meteorologia, a grande maioria dos alvos são pequenos comparados com o comprimento de onda de um radar, portanto, para o radar meteorológico, a região de Rayleigh no gráfico da figura 5 é importante. Observa-se também, no gráfico, que, quanto maior o alvo (deslocando-se para a direita no eixo horizontal), o gráfico aproxima-se ou "converge" para a unidade. Isso indica que a Área de Espalhamento σ_s aproxima-se cada vez mais do formato geométrico da esfera (observar que o eixo vertical é dado por $\frac{\sigma}{\pi R^2}$).

Portanto, a energia captada pela antena do radar é justamente a energia proveniente do espalhamento das ondas eletromagnéticas observado nestes alvos.

2.1.2 Refletividade Z

Quando a antena de radar emite pulso de onda eletromagnética, a onda viaja com a velocidade de luz e atinge vários tipos de alvos. Muitos dos alvos não são esferas, tais como aviões, pássaros, edifícios. Mas também muitos dos alvos são tipicamente pequenas gotas ou gotículas de água. Pode-se aproximar esses alvos considerando-os esferas, para efeitos de

simplificação. O fato é que bilhões dessas gotículas são atingidas pelo pulso de energia. E o efeito disso é que essas bilhões de gotículas refletirão o pulso de energia e o efeito total, ou seja, a área total de espalhamento é, matematicamente, a soma das contribuições individuais (RINEHART, 2004, p.86), equação 2.4:

$$\sigma_t = \sum_{i=1}^{n} \sigma_i \tag{2.4}$$

Pela equação 2.3, pode-se chegar a equação 2.5:

$$\sigma_t = \sum_{i=1}^{n} \frac{\pi^5 |K|^2 (2R_i)^6}{\lambda^4} \tag{2.5}$$

Chamando $2R_i = D_i$, chega-se a equação 2.6 (RINEHART, 2004, p. 92):

$$\sigma_t = \sum_{i=1}^{n} \frac{\pi^5 |K|^2 (D_i)^6}{\lambda^4} \tag{2.6}$$

Mas na equação 2.6, todos os valores são constantes, com exceção de D_i. Então:

$$\sigma_t = \frac{\pi^5 |K|^2 \sum_{i=1}^{n} (D_i)^6}{\lambda^4} \tag{2.7}$$

Define-se z como Fator Refletividade do radar pela equação 2.8 (RINEHART, 2004, p.93):

$$z = \sum_{i=1}^{n} (D_i)^6 \tag{2.8}$$

Dado que z pode ter uma grande variadade de valores, define-se o Fator Logarítmico de Refletividade de radar Z como sendo (RINEHART, 2004, p. 97):

$$Z = 10 log_{10}\Big(\frac{z}{1 mm^6/m^3}\Big) \tag{2.9}$$

onde Z é medido em decibéis relativos à refletividade de $1 mm^6/m^3$ e z é a Refletividade Linear em mm^6/m^3.

Utilizando Z, tem-se a vantagem de compressão da variação de valores possíveis. Valores de exemplo estão entre -30dBZ para neblina e +76,5dBZ para granizo intenso.

Entende-se mediante a da equação 2.8 que a energia recebida pela antena é diretamente proporcional à sexta potência do diâmetro dos vários elementos ou hidrometeoros presentes na atmosfera. Essa informação permite que, por meio do radar meteorológico, possa-se determinar a intensidade ou estimar a quantidade de hidrometeoros de determinado diâmetro, estimar os vários aglomerados desses hidrometeoros e associá-los ao nível de chuva na área de cobertura do radar.

2.1.3 Estimativa da Distância

Um dos recursos do radar é estimar a distância em que o objeto ou obstáculo se encontra. Esta estimativa é obtida pelo fato de que no espalhamento a velocidade de transmissão é a mesma da onda incidente, pois a velocidade é dependente apenas do meio de transmissão.

A velocidade das ondas eletromagnéticas é igual a velocidade da luz, que para o vácuo é 299.792.458 m/s (INMETRO, 2012), porém, quando as ondas propagam-se em meio diferente do vácuo, a velocidade é diferente, mas usualmente próxima da velocidade da luz no vácuo.

Como visto acima, a antena do radar capta o eco resultante da onda espalhada. Essa onda foi "provocada" pela energia recebida pelo pulso de ondas transmitidas pelo radar. Logo, para estimar a distância do objeto, pode-se entender que é a distância percorrida pela onda da antena para o alvo e do alvo para a antena. Conhecendo-se o tempo para esse trajeto, obtém-se, pela equação 2.10, a distância da antena ao alvo:

$$r = \frac{c_m \Delta t}{2} \tag{2.10}$$

onde :

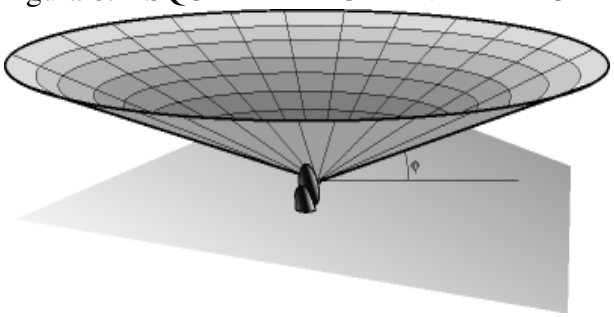
- r : distância da antena ao alvo
- c_m : velocidade da onda no meio
- Δt : tempo entre a transmissão e a recepção da onda

2.1.4 Volume de Dados

Como explicado anteriormente, o receptor do radar necessita conhecer o tempo em que o pulso de ondas é enviado e, caso alcance algum obstáculo, capte a onda ecoada. Com essa informação pode estimar a distância em que o alvo encontra-se. Para que isso ocorra, o receptor efetua chaveamentos na recepção a partir da transmissão do pulso, e dessa forma pode-se estabeler *Range Gates* ou "Porções de alcance", denominados *bins*. O chaveamento é necessário para que a antena capte apenas ecos de uma determinada distância (ou intervalo) do radar, não captando ondas fora dela. Cada bin recebe o valor da média ponderada dos alvos detectados dentro da amostra do *Range Gate*.

Para efetuar as leituras, a antena do radar usualmente efetua movimento de rotação, inclinada com ângulo em relação à horizontal denominado *Ângulo de Elevação*. Cada posição de rotação é denominado *Raio*. Em cada raio, o radar efetua a leitura nos *bins*. Usualmente o intervalo entre os raios do radar pode ser de 1°.

Figura 6: ESQUEMA DE UMA VARREDURA



FONTE: (METEOPT,)

O conjunto de leituras dos raios em uma circunferência completa é denominado *Varredura* ou *Sweep*.

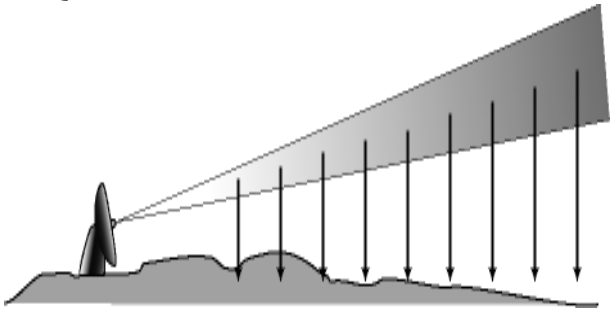
A cada *Varredura*, a antena pode ser inclinada para cima, e iniciar outro procedimento de *Varredura*. A figura 6 mostra um esquema das leituras feitas para uma Varredura.

O *Volume de Dados* do radar é o conjunto de todas as Varreduras, sendo cada varredura formada pelo conjunto de raios e cada raio efetua a leitura em *bins*. A este *Volume de Dados* dá-se o nome de *Volume Coverage Pattern* ou *VCP* .

2.2 PLAIN POSITION INDICATOR

Os dados do radar são obtidos a partir da leitura de cada *bin* em cada raio de uma varredura. A figura 8 mostra que as leituras são feitas em posições específicas, que correspondem aos *bins*.

Figura 7: ESQUEMA DE LEITURA DE BINS PARA UM RAIO



FONTE: (METEOPT,)

Plain Position Indicator (PPI) ou Indicador de Posição Plano é o mais tradicional modo de visualização de dados de radar meteorológico.

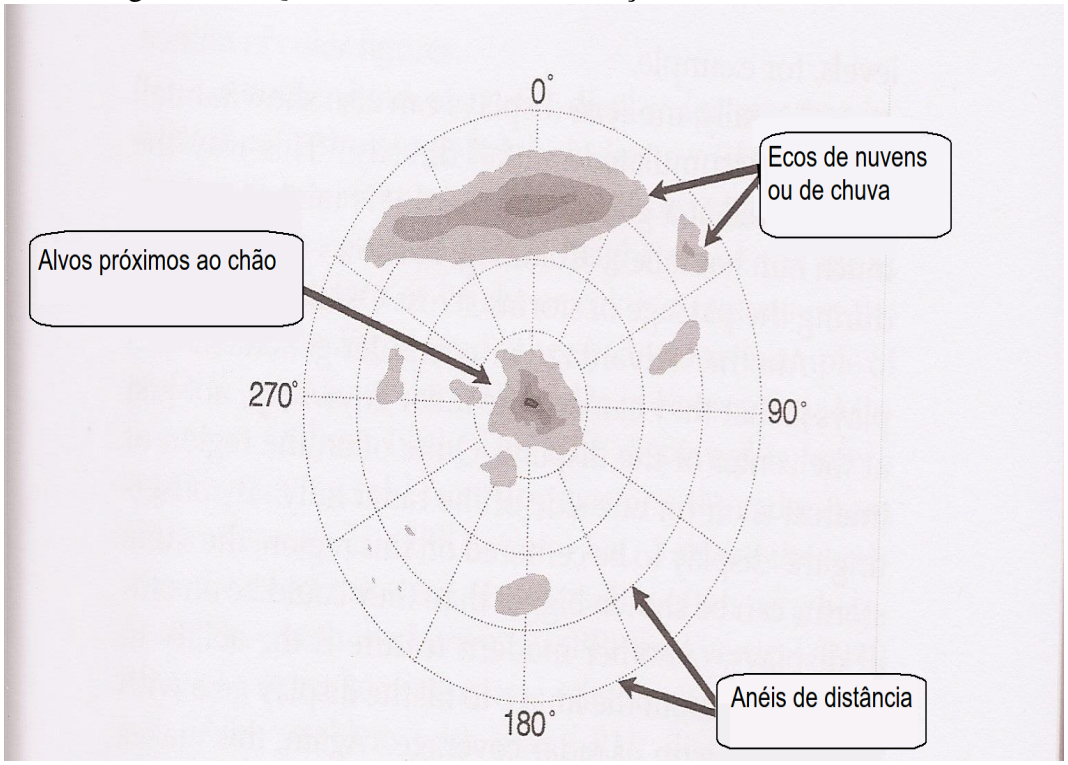
No monitor do usuário, é apresentado um desenho, usualmente incluindo um mapa,

mostrando o radar no centro. Contém ainda anéis concêntricos, onde o centro é a posição do radar. Cada anel especifica uma distância a partir do radar e desta forma o usuário pode estimar a que distância estão os ecos mostrados no monitor. Usualmente, o Norte geográfico encontra-se no topo, com Leste a direita.

Os ecos dos obstáculos são mostrados de modo que o usuário possa ter uma estimativa visual das distâncias por meio dos anéis concêntricos. Utiliza-se ainda uma tabela de cores para que o usuário possa estimar de forma visual a intensidade da refletividade sendo apresentada no monitor.

A figura 8 mostra, em forma esquemática, como é feita a apresentação em PPI.

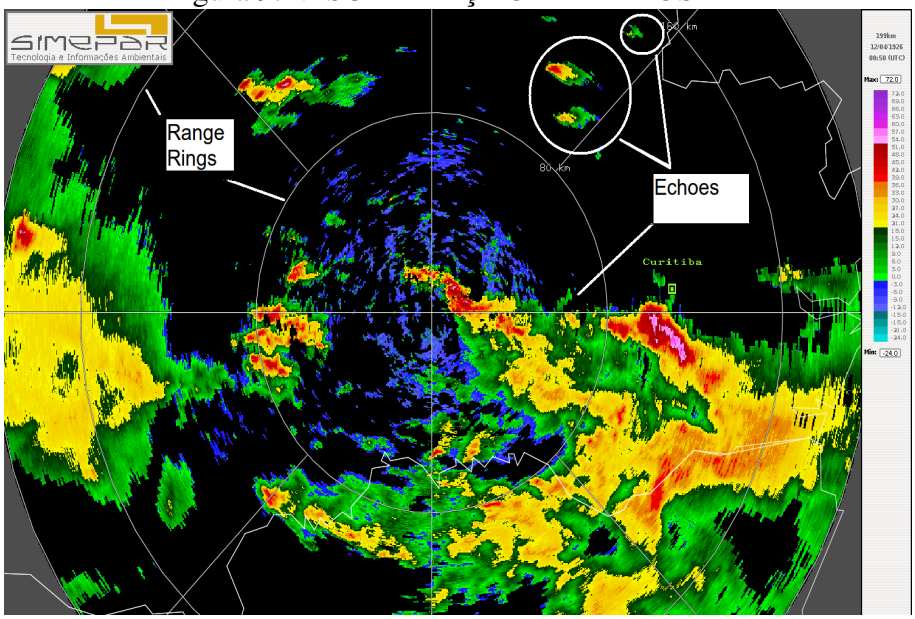
Figura 8: ESQUEMA DE APRESENTAÇÃO DE DADOS EM PPI



FONTE: (RINEHART, 2004, p. 45)

A figura 9 mostra um exemplo de visualização real de dados em PPI por meio de *software* desenvolvido internamente no Simepar.

Figura 9: VISUALIZAÇÃO DE DADOS PPI



FONTE: Simepar

2.3 RADAR METEOROLÓGICO DO SIMEPAR

O Instituto Meteorológico Simepar (www.simepar.br) possui um radar para observação de eventos meteorológicos situado na região central do estado, no município de Teixeira Soares, latitude -25,505313° e longitude -50.361330°, na altitude de 1016 m. O radar é um equipamento Banda S Dual Doppler, modelo DWSR-95S da EEC Corporation, montado sobre uma torre de concreto de 30 m. O conjunto da antena do radar mede 8,2 m de diâmetro e está protegido por uma cúpula, denominada *Radome*. A figura 10 mostra o radar e a antena.

A antena gera pulsos com abertura de 0,9° e efetua sequência pré definida de varreduras de 360° cada, com diferentes elevações. Os alcances operacionais são de 200 km e 480km. Efetua, então varreduras com esses alcances e com variação nas elevações, produzindo um volume completo de dados a cada 12 minutos aproximadamente. Atualmente, é utilizado um software desenvolvido pela equipe de desenvolvimento do próprio Simepar que apresenta os dados para análise e visualização.

O sistema está configurado para efetuar 14 varreduras com raio de 200 km. Cada varredura com 360°, ou seja, 360 raios, com 800 *bins* cada raio, totalizando 288.000 valores. No total das 14 varreduras, 4.032..000 valores são obtidos.

Figura 10: RADAR METEOROLÓGICO DO SIMEPAR - TORRE E ANTENA (DENTRO DA CÚPULA DE PROTEÇÃO

FONTE: Simepar

Figura 11: RADAR METEOROLÓGICO DO SIMEPAR - DETALHE DA ANTENA

FONTE: Simepar

3 WEBGL

WebGL, como descrito pelos próprios criadores (KhronosGroup Inc, 2011), é um padrão para a WEB, para definição de API de modo que possa servir de acesso de baixo nível para gráficos 3D e disponível por meio do elemento *canvas* do HTML Especificação 5 (ou simplesmente HTML 5)

Baseada em OpenGL ES SL, WebGL permite a criação de *shaders* utilizando OpenGL ES, trazendo o desenvolvimento de aplicações gráficas em 2D e 3D utilizando JavaScript diretamente no navegador, sem necessidade de nenhum tipo de acessório ou *plugin*. Até o momento da escrita deste trabalho, apenas os navegadores Mozilla© Firefox e Google© Chrome têm suporte nativo ao WebGL. O navegador Apple© Safari tem suporte parcial.

Juntamente com as tecnologias de HTML5 e Javascript, WebGL tem permitido que o desenvolvimento de aplicações, antes imaginadas em um contexto de criação de programas compilados [1], ou seja, programação convencional que gera programas em código de máquina (ou binário), possam fazer parte do contexto de Internet. Essas novas tecnologias em conjunto estão permitindo a portabilidade [2] de jogos que demandam alto desempenho da placa gráfica para a web, conforme menciona (ANTTONEN et al., 2011) citando o exemplo do jogo popular *Quake 2* [3]. A figura 12 mostra uma imagem do vídeo de demonstração.

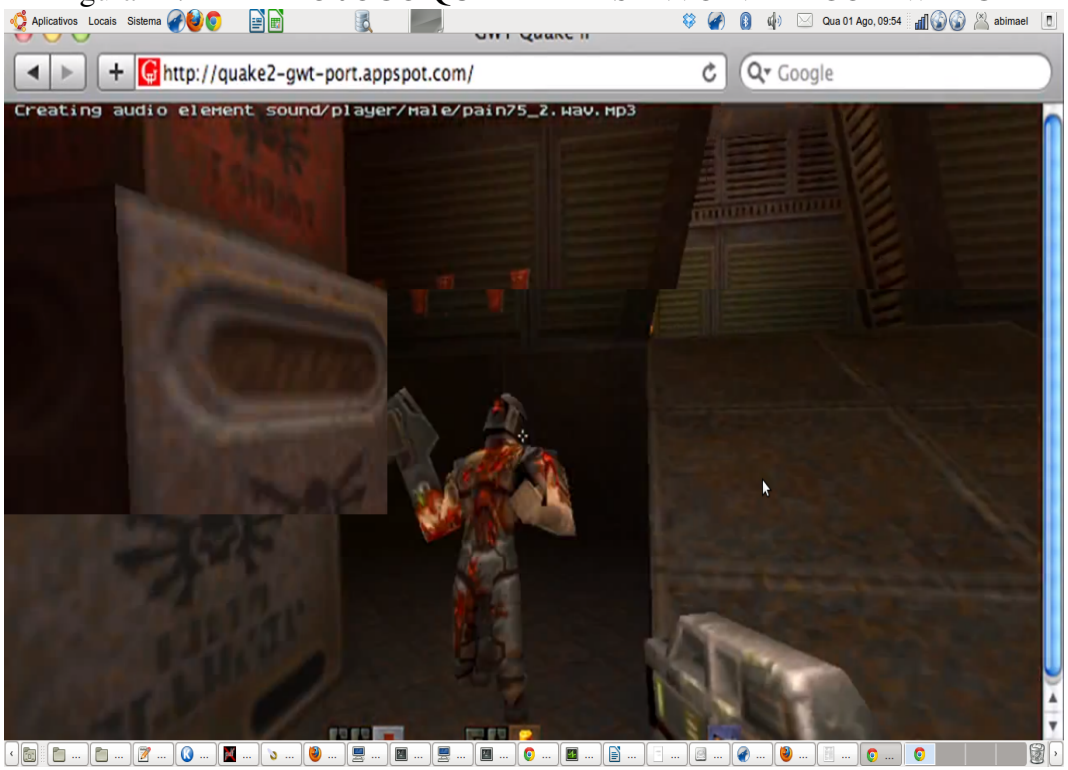
Percebe-se então que se torna possível disponibilizar elementos de computação gráfica diretamente no navegador utilizando recursos sofisticados que podem ser aproveitados inclusive por jogos, demandando alto desempenho de placas gráficas. Contudo, a implementação de jogos, apesar dos requisitos de desempenho devido às demandas dinâmicas e a interatividade que o jogador espera, implica muitas simplificações no ambiente, que não podem ser simplificadas

[1] Compilar é o processo pelo qual o compilador de determinada linguagem de programação converte o texto em código ou linguagem de máquina, permitindo assim ser executado pelo computador.

[2] Portar tem o sentido de permitir a execução de um determinado *software* para outra plataforma ou sistema operacional. Normalmente este processo implica em, pelo menos, recompilação do código fonte para a outra plataforma ou reescrita de código fonte.

[3] Este jogo foi muito popular na década de 90 e na primeira década do século 20. Seu engine trouxe grandes avanços para o desenvolvimento de jogos. Permitia programação, o que popularizou a alteração feita pelos fãs do jogo e implementando "mods" (modificações). Em 1999, o código fonte foi disponibilizado sobre licença GPL. O jogo em sua versão utilizando WebGL encontra-se disponível em http://playwebgl.com/games/quake-2-webgl/

Figura 12: TELA DO JOGO QUAKE 2 DESENVOLVIDA COM WEBGL



FONTE: http://code.google.com/p/quake2-gwt-port/

extraído do filme disponibilizado no site.

quando da visualização de dados. Neste trabalho, o protótipo desenvolvido visa a possibilidade de visualização de dados em ambiente Web, utilizando esta promissora tecnologia.

3.1 Histórico

Segundo (ROST et al., 2006, p. 1), OpenGL é uma API padrão da indústria, finalizada em 1992 e com a primeira implementação disponibilizada em 1993. Era compatível com a API denominada IRIS GL, de propriedade da Silicon Graphics, Inc ©. Visando estabelecer um padrão para o desenvolvimento, Silicon Graphics, em colaboração com outras empresas de *hardware* criou um padrão chamado OpenGL.

Para controle da evolução, foi criado pela Silicon Graphics *Architecture Review Board* (ARB) [4] que era formado por um conjunto de companhias tais como Apple©, IBM©, Microsoft© e Intel© entre outras.

A diretiva principal da criação do OpenGL foi manter um único padrão que provesse acesso as capacidades de *hardware* até ao mais baixo nível possível e ainda permitir inde-

[4] Grupo de Revisão da Arquitetura. O termo em inglês foi mantido por ser muito utilizado na literatura

pendência de fabricantes de *hardware*. Atualmente OpenGL encontra-se na versão 4.2 (Khronos-Group Inc, 2012) .

O projeto e especificação inicial de OpenGL contemplava funcionalidades especificadas por meio de funções fixas. Porém, com novas funcionalidades disponíveis no *hardware*, foram sendo implementadas Extensões que permitiam modificações na OpenGL. As Extensões ao OpenGL permitiam adaptar a biblioteca para que pudesse utilizar funcionalidades específicas de uma placa gráfica. Com o lançamento da versão 2.0, o *pipeline* [5] da OpenGL foi alterado para permitir que funcionalidades pudessem ser programadas. Dessa forma, o processamento de Vértices e processamento de Fragmentos foi aberto para controle do desenvolvedor por meio de programação de *shaders*. A Linguagem de programação desenvolvida para programação dos *Shaders* de vértices e de fragmento é a *OpenGL Shading Language* (OpenGL SL).

Dessa forma, juntamente com DirectX [6], OpenGL tornou-se padrão da indústria de desenvolvimento de Computação Gráfica. Sua característica de ser multiplataforma [7] permitiu seu uso em diversos sistemas e ambientes, tais como desenvolvimento de aplicações de CAD , processamento de vídeo, animação e visualização científica (MUNSHI; GINSBURG; SHREINER, 2009, p. 1).

Com o advento de novas tecnologias, além do desenvolvimento de OpenGL no ambiente *desktop*,[8] houve necessidade de desenvolvimento de API para ambientes móveis, tipicamente para telefones e outros dispositivos, e mais recentemente, tablets. Dessa forma, foi criada uma nova API denominada OpenGL ES, e segundo (MUNSHI; GINSBURG; SHREINER, 2009, p. 1), essa API tinha como missão contemplar as limitações de hardware desses novos dispositivos.

Segundo (BLYTHE, 2002, p. 1), a especificação 1.0 da OpenGL ES foi desenvolvida a partir da versão 1.3 da OpenGL e segundo (MUNSHI; GINSBURG; SHREINER, 2009, p. 2), a especificação 1.1 foi baseada na versão 1.5 da OpenGL. Ambas as especificações eram basedas em versões da OpenGL que tinham suas funcionalidades fixas. Porém, a partir da versão 2.0 da OpenGL ES, que foi baseada na versão 2.0 da OpenGL, a programação de *Shaders* foi

[5] Sequência de unidades funcionais, onde cada etapa fornece um resultado que pode ser utilizado por outra unidade.

[6] DirectX é a API e conjunto de bibliotecas gráficas desenvolvidas pela Microsoft para uso apenas em sistemas Windows©e para o console de jogos XBOX©

[7] Multiplataforma é *software* que pode ser executado em ambientes ou sistemas operacionais diferentes, tais como Linux, Windows, Unix e MacOS. Usualmente necessita de compilação específica para cada plataforma, porém é diferente de *software* desenvolvido para apenas uma única plataforma

[8] desktop é expressão utilizada para computadores de mesa ou computadores portáteis (notebooks) que utilizam configurações similares a computadores de mesa

Figura 13: FIGURA DE JOGO DESENVOLVIDO COM OPENGL ES

FONTE: (MUNSHI; GINSBURG; SHREINER, 2009, Color Plate 1)

adicionada pela especificação de linguagem de programação específica denominada OpenGL ES *Shading Language*.

OpenGL e OpenGL ES são APIs para computadores *desktop* e dispositivos móveis respectivamente, porém, percebeu-se que havia necessidade de padrões ou API para computação gráfica disponível para a Internet. Na tentativa de desenvolvimento de aplicativos que pudessem efetuar a renderização [9] no ambiente Internet foram desenvolvidos *plugins*, que pudessem ser instalados no navegador e não nativamente pelo próprio navegador (GRACIA JAVIER ; BAYO, 2012).

A primeira tentativa para criação de um padrão para disponibilizar aplicações de Computação Gráfica na Internet foi o desenvolvimento da VRML , proposto por Ragget (RAGGETT, 1994), que estabeleceu um método para renderização de dados 3D utilizando marcações. Para que renderização tivesse efeito, havia necessidade de instalação de um *plugin*, como o popular Cortona VRML Viewer Client.

O VRML foi então substituído pelo X3D desenvolvido pelo Web3D Consortium (WEB3D Consortium, 2012) (http://www.web3d.org). Mesmo sendo um padrão para definição da geome-

[9] Renderização é termo derivado do verbo da língua inglesa “to render” que significa “apresentar as imagens no monitor de vídeo” (RENDER,). Termo foi mantido por ser termo comum e amplamente utilizado na literatura.

tria em XML [10] , a renderização do conteúdo dava-se por meio de *plugins* ou de *applets* Java. Os dados contidos dentro do documento HTML era renderizado por *applet* por meio da Máquina Virtual Java ou *JVM*. Atualmente, *JVMs* estão disponíveis em vários sistemas operacionais, permitindo que aplicações sejam independentes de plataforma (GRACIA JAVIER ; BAYO, 2012, p.5).

Mesmo com os desenvolvimentos de novas aplicações na tentativa de utilizar o ambiente OpenGL através de códigos Java, ainda assim havia o problema de que a máquina virtual JVM é uma camada que não acessa o *hardware* diretamente. Além disso, o navegador era simplesmente o meio para efetuar o *download* automático do *byte code* para *applet* ou *plugin*, mas o responsável pela renderização era o *applet* (GRACIA JAVIER ; BAYO, 2012, p. 5).

WebGL então vem justamente suprir essa lacuna, tornando as aplicações independentes de *plugins* ou *applets* para renderização e eliminando a necessidade de uma camada a mais para acesso ao *hardware*, bem como da necessidade de quaisquer instalações de códigos ou ambientes de terceiros (*plugins*).

3.2 PIPELINE DE APLICAÇÃO WEBGL

A figura 14 mostra o pipeline de aplicação desenvolvida em WebGL mostrando a comunicação entre ela (codificada em Javascript) e os principais elementos da WebGL que conduzem à renderização das imagens pela API WebGL.

3.2.1 Aplicação Web

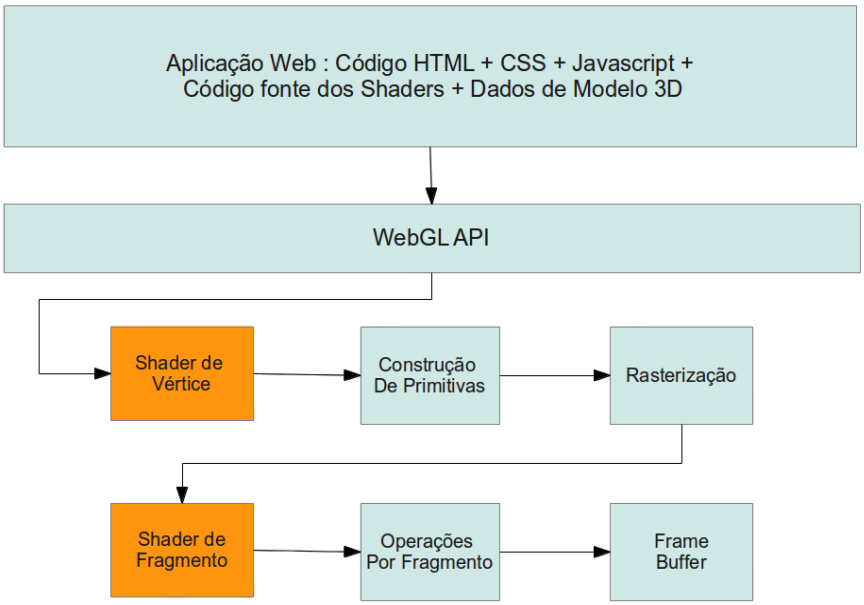
A aplicação Web é composta de vários elementos:

- Código HTML ou tags HTML: são os elementos destinados à construção da página em si. O código HTML contém o elemento *canvas* que define o ambiente WebGL;
- CSS [11] Folhas de estilo CSS para definir a aparência dos elementos HTML ;
- Código Javascript que:
  - proverá os dados dos modelos;

[10] XML: Extensible Markup Language ou Linguagem de marcação extensível. Define conjunto de regras para definição de dados, atributos e valores, por meio da codificação de dados, principalmente para troca de informações entre diferentes aplicações.

[11] Cascading Style Sheet ou Folha de estilos

Figura 14: PIPELINE DE APLICAÇÃO DESENVOLVIDA COM WEBGL



FONTE: (ANYURU, 2012, p. 8) adaptado pelo autor

  - responderá aos eventos que possam ser disparados para o correto funcionamento da aplicação.

- O código fonte dos *shaders*, inserido em tags específicas.

3.2.2 WebGL API

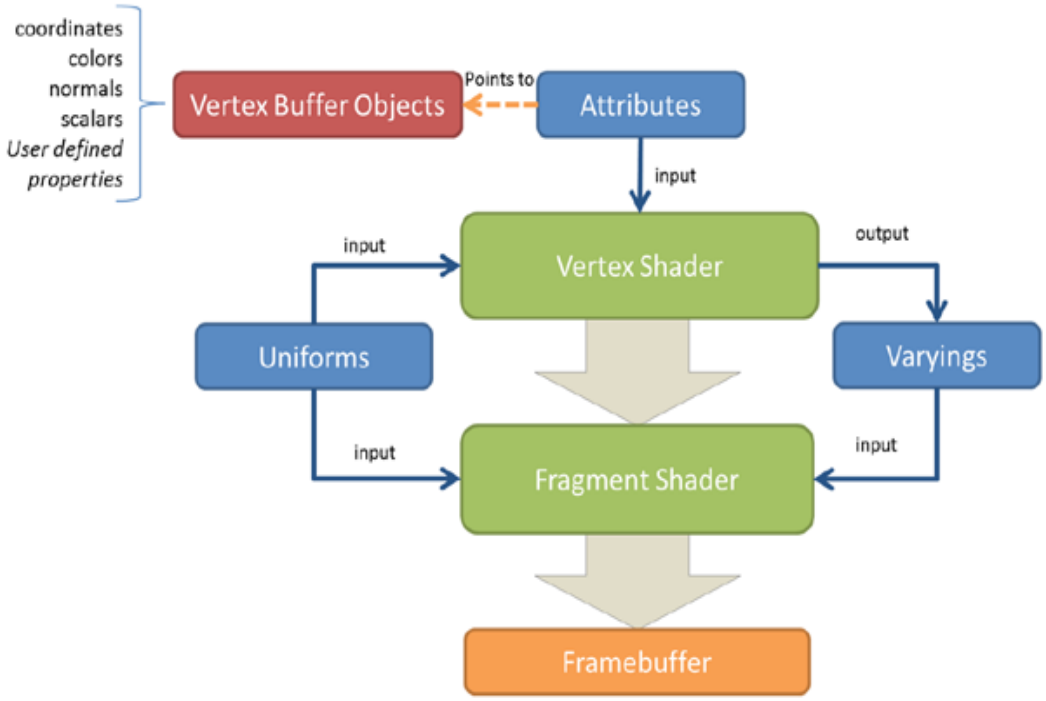
A API WebGL provê a comunicação entre a camada HTML e o código Javascript, permitindo que a execução dos *shaders* efetuem a renderização dos objetos modelados e contidos na Aplicação Web.

O contexto WebGL é obtido a partir da *tag canvas* e operações da API são invocadas a partir desse contexto.

3.2.3 Shader de Vértices

Shader de Vértices ou *Vertex Shader* é, segundo (MUNSHI; GINSBURG; SHREINER, 2009, p.4), o *shader* que executa código de propósito geral para manipulação dos vértices. É invocado para cada vértice e manipula dados por vértice (CANTOR; JONES, 2012, p.25).

Figura 15: DETALHES DA API WEBGL



FONTE: (CANTOR; JONES, 2012, p. 24) adaptado pelo autor

A figura 15 mostra esquematicamente como os dados são passados ao *shader* de Vértices :

- Código fonte do *shader* escrito em OpenGL ES SL;
- Dados dos vértices providos pelo código javascript usando matrizes de vértices (*vertex arrays*), denominados *Vertex Buffer Objects (VBOs)* [12] utilizando *Atributes*;
- *Uniforms* - Dados constantes para todos os vértices e que são utilizados pelo *shader*.

O código fonte do *shader* é escrito com OpenGL ES SL dentro de tags `<script>` onde o atributo *type* contendo o valor `type="x-shader/x-vertex"`.

Os *Atributes*[13] são tipos de variáveis que "apontam" para os *Vertex Buffer Object* ou VBOs. É por meio dos *Atributes* que o código WebGL tem acesso aos dados definidos ou contidos nos VBOs. Em cada Ciclo de Renderização, os valores dos atributos são diferentes e executam o código do Shader de Vértice em paralelo na GPU.

[12] Vertex Buffer Objects ou VBOs são objetos instanciados pelo código Javascript e que contém as coordenadas de vértices, valores de vetores normais, valores de cores, escalares e, dados específicos do usuário. São definidos como matrizes e definem uma área de memória de uso específico pelo *shader* de vértices

[13] Attributes são traduzidos como atributos

Os *Uniforms* são variáveis de entrada para ambos os *shaders* de vértices e fragmento. Segundo (CANTOR; JONES, 2012, p.27), diferentemente dos *Atributes*, os *Uniforms* são constantes durante todo o Ciclo de Renderização, ou seja, enquanto os *Attributes* são diferentes para cada instância do *shader* de vértice, os valores do *Uniforms* são os mesmos para todas as instâncias tanto do *shader* de vértice quanto para o de fragmento (CANTOR; JONES, 2012, p.26). Um exemplo típico é a posição de uma fonte de luz, que não muda tanto para o *shader* de vértices quanto para o de fragmento e, portanto, deve ser definido em uma variável do tipo *Uniform*.

Quando é necessário passar dados do *shader* de vértice para o *shader* de fragmento, utilizam-se variáveis definidas como *Varyings*. Uma variável definida como *Varying* no *shader* de vértice, pode ser acessada pelo *shader* de fragmento com o mesmo nome. Dessa forma, valores calculados ou definidos no *shader* de vértice são aproveitadas no *shader* de fragmento.

3.2.4 Construção de Primitivas

Conforme (ANYURU, 2012, p.12), o pipeline WebGL precisa montar os vértices já devidamente disponibilizados pelo *shader* de vértices em primitivas geométricas individuais, tais como linhas, pontos e triângulos. Para cada elemento geométrico, decide-se se ele está ou não está dentro da região de visualização 3D. A decisão é tomada se o elemento está contido no Tronco de visão:

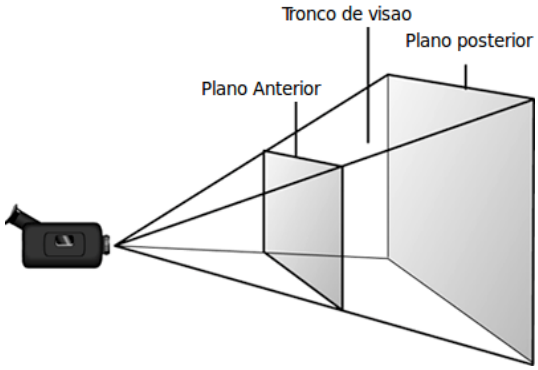
- Se o elemento está contido, ele segue para o próximo estágio do Pipeline;
- Se o elemento está totalmente fora, ele é descartado;
- Se o elemento está parcialmente contido, ele é "cortado" (clipping), de modo que apenas a parte que está contida siga para o próximo estágio.

Na figura 16 é descrito o Volume de Visão (ou Tronco de visão), mostrando o volume onde os elementos devem estar contidos para serem construídos.

3.2.5 Rasterização

Após a construção e definição das primitivas que devem ser renderizadas, elas são decompostas em partes menores denominadas *fragmentos*. Os *fragmentos* são, na verdade, os *pixels* que serão enviados ao *shader* de fragmento.

Figura 16: VOLUME DE VISÃO

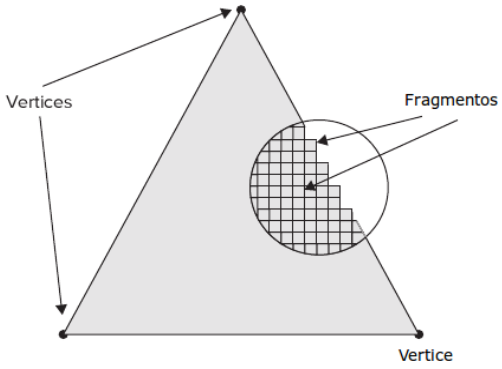


FONTE: (CANTOR; JONES, 2012, p. 25) adaptado pelo autor

Um exemplo simples de como a Rasterização funciona é o de uma linha que é formada por dois vértices e que cobre dez pixels entre os vértices. No processo de rasterização, esta linha é dividida em dez pedaços, um pedaço para cada fragmento.

A figura 17 mostra como é o processo de rasterização, decompondo em fragmentos a partir de três vértices para preenchimento da superfície.

Figura 17: EXEMPLO DE FRAGMENTO



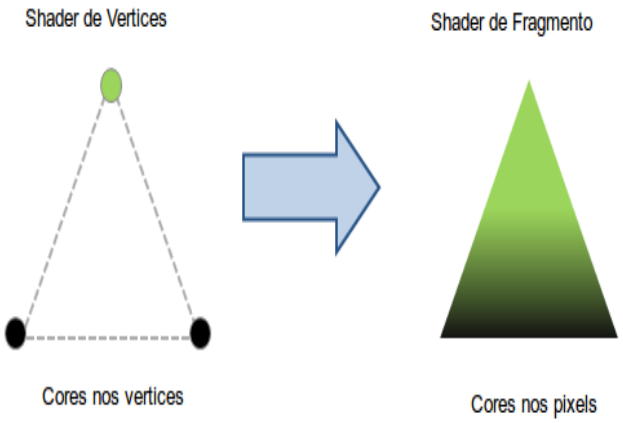
FONTE: (ANYURU, 2012, p.16) adaptado pelo autor

Para cada *pixel* ou *fragmento* são definidos cor, coordenadas de textura e todos os outros elementos necessários para renderização. Os valores são determinados pela interpolação a partir dos valores dos vértices (ROST et al., 2006, p.15).

3.2.6 Shader de Fragmento

O outro elemento programável do Pipeline WebGL é o *shader* de fragmento.

Figura 18: DIFERENÇA ENTRE SHADER DE VÉRTICES E DE FRAGMENTOS



FONTE: http://flylib.com/books/en/2.940.1.39/1/ - adaptado pelo autor

O *shader* de fragmento atua sobre os fragmentos e principalmente deve definir ou calcular a cor de cada fragmento presente entre os vértices, conforme mostrado na figura 17. A figura 18 mostra um exemplo da definição de cores estabelecidas pelo *shader* de fragmento. Basicamente cada fragmento é um pixel, mas evita-se utilizar o termo pixel, porque durante as operações sobre os fragmentos, alguns poderão ser descartados e não serão renderizados. Logo, apenas fragmentos que serão renderizados recebem o nome de pixel.

A figura 15 mostra como valores são passados para o *shader* de fragmento :

- Por meio de *Uniforms*, valores definidos pelo usuário/aplicação;
- Por meio de *Varyings*, com valores obtidos ou definidos no *shader* de vértices.

O código fonte do *shader* de fragmentos é escrito com OpenGL ES SL dentro de *tags* `<script>` onde o atributo type contendo o valor `type="x-shader/x-fragment"`.

3.3 WEBGL E VISUALIZAÇÃO CIENTÍFICA

A utilização de WebGL no contexto científico está em fase inicial, visto que a tecnologia é recente. Percebe-se entretanto, que alguns artigos científicos mostram o potencial de uso desta nova tecnologia.

Fazendo uma pesquisa no site de Periódicos da Capes (CAPES, 2012), constatou-se:

- Utilizando “busca por assunto”, contendo “WebGL” no Título, marcando “Expandir meus resultados”, obtiveram-se os seguintes resultados :

- Refinando pelo Tópico “WebGL” :

Coleção	Quantidade	Ano	
ACM	10	2010	4
		2011	5
		2012	1
IEEE	10	2011	9
		2012	1
Directory of Open Access Journals	2	2011	1
		2012	1
SciVerse Science Direct (Elsevier)	1	2012	1
Total	23		

- Refinando por “Three Dimensional Displays”, obtiveram-se seis artigos publicados;

Atas de Congresso	6	2011	6
Total	6		

- Refinando por “HTML5”:

Atas de Congresso	6	2011	5
		2012	1
Total	6		

Entende-se com os números dos resultados obtidos pela pesquisa no site de periódicos da Capes que o uso desta nova tecnologia estão em fase inicial. Durante a execução deste trabalho, o ano de 2011 foi o que apresentou maior número de trabalhos listados, demonstrando que já há uma busca por aplicações de Visualização Científica que utilizam a tecnologia para apresentação de seus dados.

3.4 CONSIDERAÇÕES FINAIS

O desenvolvimento de aplicações para computação gráfica, jogos, e Visualização Científica entram em novo patamar e paradigma com o advento da WebGL. Agora, o navegador

de internet passa a conter a infraestrutura necessária para disponibilizar a aplicação, sem necessidade de *plugins* ou qualquer outro elemento para suporte. Em contrapartida, o código deve ser escrito em JavaScript e será interpretado em tempo de execução, ou seja, não há código compilado em linguagem específica para o computador. Dessa forma, a execução da aplicação pode ser comprometida pelo fato de Javascript ser interpretado, mas por outro lado, existem várias formas de prover os dados, inclusive remotamente. Percebe-se que WebGL pode acarretar uma revolução e novas oportunidades para desenvolvimento de aplicações.

No contexto científico, percebe-se que trabalhos visando ao uso desta nova tecnologia estão apenas iniciando, visto que a tecnologia é recente, mas o potencial para desenvolvimento de novas aplicações é grande.

4 IMPLEMENTAÇÃO DO VISUALIZADOR

A implementação do visualizador proposto neste trabalho transcorreu de modo que o resultado final atendesse ao objetivo de que a ferramenta pudesse renderizar as imagens em tempo hábil e compatível com ambiente operacional de trabalho do meteorologista na Internet, utilizando WebGL.

O desenvolvimento de visualizadores de dados para radar meteorológico em 3D não é um assunto por si só inédito. Outras ferramentas já foram desenvolvidas como em (ERNVIK, 2002). Contudo, as aplicações usualmente são desenvolvidas utilizando bibliotecas e linguagem C, que compilam e geram programas executáveis para uma plataforma específica. Por exemplo, programas que executam apenas em sistemas operacionais Windows©. Esse tipo de desenvolvimento implica que não se pode ter aplicações fáceis de serem portadas para outros ambientes e mesmo não são desenvolvidas tendo o ambiente da Internet como ambiente de execução. Neste trabalho, foca-se no desenvolvimento de visualizador que possa ser utilizado na Internet sem uso de *plugins* ou qualquer outro software adicional além do navegador com suporte a WebGL.

Apesar de trabalhos sobre uso de outras técnicas para apresentação dos dados utilizando Reconstrução de superfícies, como em (CESAR JR, 1994) e também o uso de criação de texturas, visando uma "falsa" impressão de dados tridimensionais como em (RU, 2007) implicam tempo para geração das imagens de 30 minutos (RU, 2007, p.44), o que é inviável em ambiente operacional, ou seja, não alcança o objetivo deste trabalho. Ainda, em (DOGGETT IV et al., 2002), utiliza-se Matlab© para gerar as imagens, o que implica que o meteorologista trabalhe com os dados e utiliza essa ferramenta para apresentação e visualização. São operações que não são úteis nas atividades operacionais.

Percebe-se que técnicas que podem produzir imagens apreciadas do ponto de vista estético não são adequadas para uso em ambiente operacional. Dessa forma, opta-se neste trabalho por técnicas básicas, porém visando a apresentação da visualização dos dados de forma prática, bem como em ambiente de Internet.

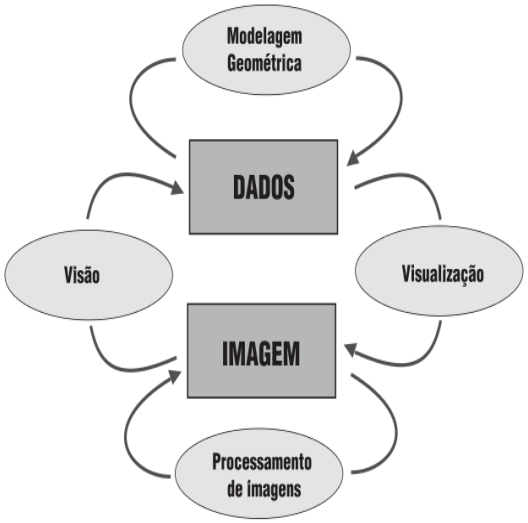
Esse capítulo tem como objetivo apresentar a Metodologia e descrever a Implementação do protótipo do visualizador, utilizando WebGL.

4.1 METODOLOGIA

Segundo a taxonomia da Computação Gráfica proposta por (GOMES; VELHO, 2008, p.2), a Modelagem Geométrica trata de descrever e estruturar dados geométricos no computador, enquanto que a Visualização (ou Síntese de Imagens) gera a imagem a partir desse Modelo Geométrico. Ainda dentro desta taxonomia, processamento de imagens, trata uma imagem uma imagem e produz outra imagem como saída. A Visão Computacional tem como objetivo obter informações geométricas, topológicas ou físicas a partir das imagens de entrada (GOMES; VELHO, 2008).

A figura 19 mostra esquematicamente a Taxonomia da Computação Gráfica, segundo (GOMES; VELHO, 2008).

Figura 19: TAXONOMIA DA COMPUTAÇÃO GRÁFICA



FONTE: (GOMES; VELHO, 2008, p. 2)

Um dos propósitos da Visualização, segundo (TELEA, 2008, p.8), é o de obter a compreensão por meio de gráficos e imagens que sejam interativos, relacionados com processos tais como simulações ou algum processo físico do mundo real. Entende-se por um sistema interativo, que a partir dos dados fornecidos por modelos ou fornecidos por algum dispositivo de medida, esse sistema apresenta a imagem ao usuário reconstruída a partir dos dados fornecidos pelo dispositivo de medida. O usuário, por sua vez, interage com esta aplicação e com sua

observação, tem a compreensão do fenômeno modelado ou medido (TELEA, 2008, p.8,9).

O processo de visualização pode ser resumido, segundo (TELEA, 2008, p.37) por:

- Aquisição de dados de interesse em um conjunto de dados (dataset) discreto;
- Mapear o conjunto de dados em primitivas gráficas;
- renderizar as primitivas gráficas para obter a imagem desejada.

A Aquisição de dados é feita por meio de um sensor ou equipamento de medida, ou ainda, um modelo numérico. A aquisição de dados normalmente fornece uma quantidade de dados superior ao que é possível apresentar pelo sistema de visualização, pois os dados são obtidos de forma contínua, enquanto que o equipamento que reconstruíra a imagem necessita de dados em um domínio específico. Desta forma, os dados precisam ser discretizados em um domínio finito. Nesta operação, deve-se definir não somente os pontos a serem visualizados nesse domínio finito, mas bem como a interpolação entre os pontos da discretização. Os pontos de discretização são chamados *sample points* ou *pontos de amostragem* e o conjunto desses pontos, bem como os valores de limites (máximo e mínimo) são denominados *Dataset* [1] (TELEA, 2008, p.40,41).

O Mapeamento dos dados em primitivas gráficas consiste em, a partir de seus valores, associá-los ao elemento geométrico destinado para representar a informação. O mapeamento é feito pela definição de células, que são também escolhidas em função da escolha do tipo de *Grid* (ou Grade). Os principais tipos de células são ((TELEA, 2008, p.54):

- Vértice
- Linha
- Triângulo
- Retângulo
- Tetraedro
- Hexaedro
- Paralelepípedo

[1] Dataset significa literalmente conjunto de dados; o termo é utilizado aqui para descrever o conjunto de dados geométricos e que contém a informação propriamente definida pelos dados a serem observados

- Pirâmide
- Prisma

Os tipos de Grades básicos são :

- Grade Uniforme
- Grade Retilínea
- Grade Estruturada
- Grade Não-estruturada

Neste trabalho, entendeu-se que se devia adotar uma implementação “conservadora” para verificar se a quantidade de dados poderia ser apresentada de modo que o processo de renderização pudesse ser efetuado em tempo hábil e que pudesse ser disponibilizado em ambiente operacional, conforme o primeiro objetivo específico. O uso de técnicas de renderização de volume poderiam levar a um tempo muito grande para apresentar as imagens, causando algum tipo de desmotivação ao usuário. Dessa forma, tomaram-se as seguintes decisões quanto à geometria da grade de apresentação:

- Uso de célula do tipo Vértice, onde cada *Bin* está em uma célula;
- Uso de grade Estruturada;

A célula de tipo Vértice é uma célula que representa um ponto. Optou-se por utilizar o ponto para que a renderização fosse efetuada rapidamente e acarretasse menor custo computacional por parte da placa gráfica (GPU). Dessa forma, cada *bin* é posicionado geometricamente sobre um vértice para representação dos dados na grade.

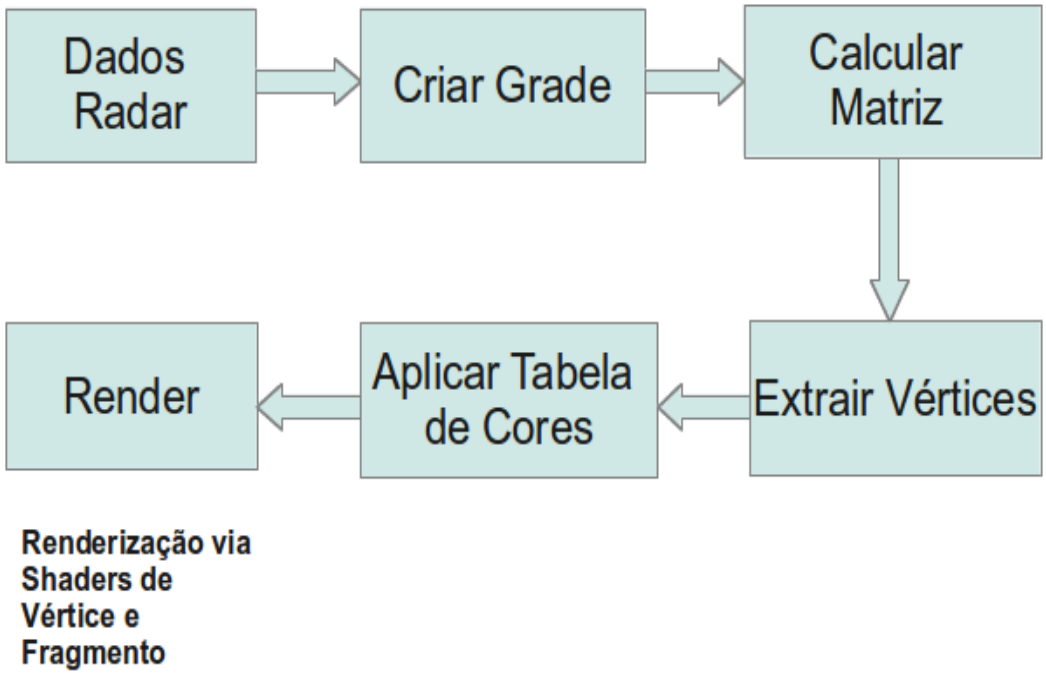
A grade Estruturada foi escolhida em função de que os dados estão em coordenadas esféricas. Dados que estão em geometria ou coordenadas retilíneas podem ser inseridos em grade do tipo **Uniforme** ou **Retilíneo**. O sistema de coordenadas adotado foi o de coordenadas esféricas em três dimensões.

Visto que a refletividade é uma grandeza escalar, medida e informada pelo *hardware* do radar meteorológico, pode ser apresentada apenas por mapeamento de cores (ou tabela de cores), visto que é a forma mais comum de representação desse tipo de dados, conforme (TELEA, 2008, p.129).

4.2 IMPLEMENTAÇÃO

O principal objetivo deste trabalho é de prover uma ferramenta de visualização de dados de radar meteorológico em ambiente Web. Para isso,há necessidade de extrair os dados brutos fornecidos pelo conjunto hardware-software do radar e, a partir dos dados extraídos, fornecer estrutura de dados disponível ao navegador de modo que o contexto WebGL possa renderizar a imagem correspondente aos dados.

Figura 20: PIPELINE DO VISUALIZADOR



FONTE: o Autor

A figura 20 mostra o Pipeline do Visualizador.

A aplicação é dividida em 2 partes, pois os dados do radar são fornecidos por programa em Java, que entrega dados para uma aplicação Javascript que é executada no navegador Internet compatível com WebGL.

O código desenvolvido em Javascript é dividido em :

- Código da Aplicação;
- Código de Biblioteca utilitária.

O Código da Aplicação é o código que está “embutido” na página da aplicação e é responsável por:

- Inicialização de objetos;
- Inicialização do contexto WebGL;
- Preenchimento dos dados;
- Renderização.

O Código da Biblioteca Utilitária está em arquivo separado da página html e é responsável por:

- Conter funções úteis;
- Definição das classes de matrizes e vetores;
- Definição da Classe WebGL;
- Definição da Classe RadarGridRender;
- Definição da Classe StructuredGrid.

A figura 21 mostra o Diagrama de Classes UML [2] das Classes que definem o conjunto de objetos que serão responsáveis pela renderização dos dados.

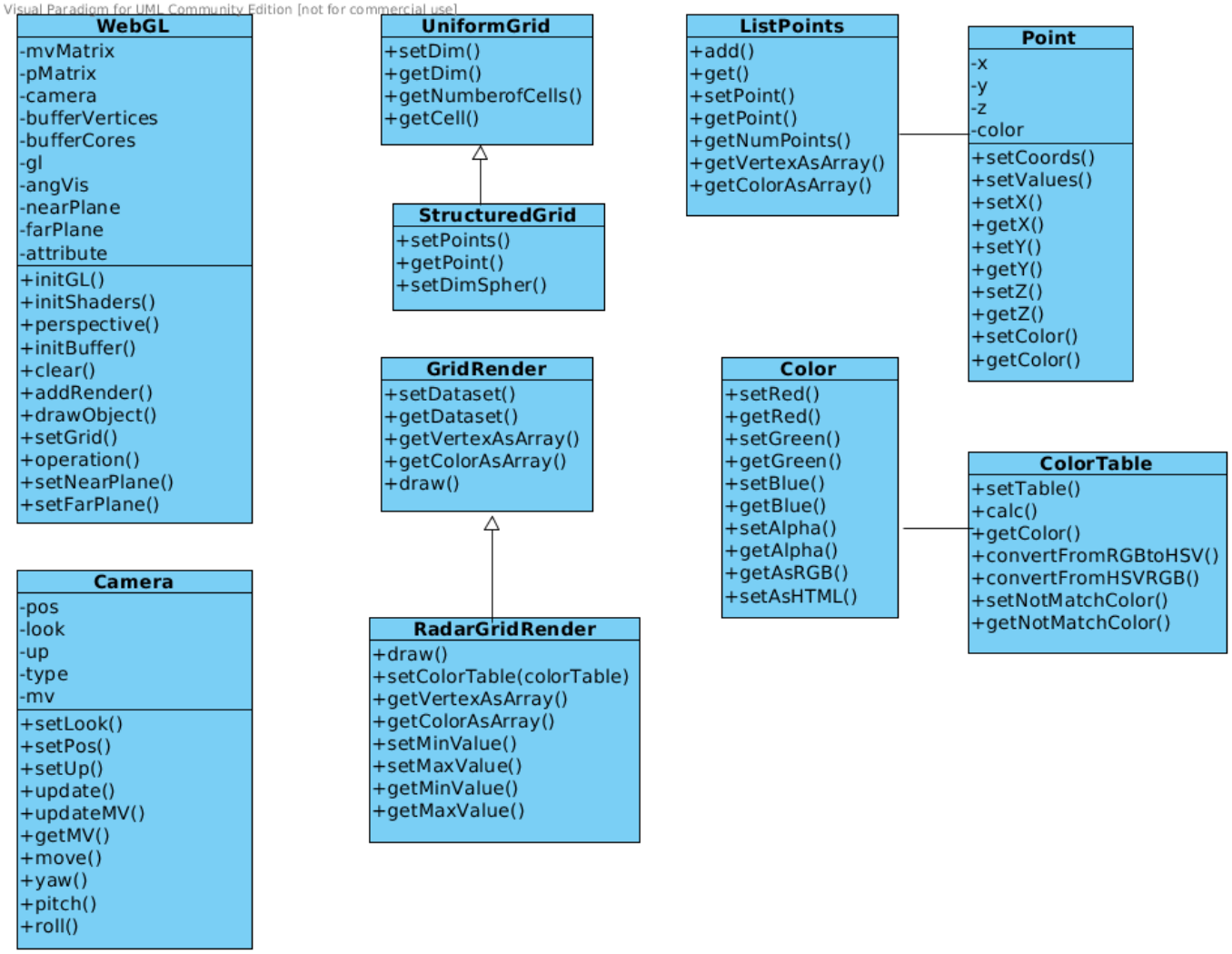
Classe WebGL Classe responsável por conter o contexto WebGL da página HTML e as definições de câmera, *buffer* de vértices e cores. Inicializa o contexto webGL, *shaders* e os *buffers*.

Classe UniformGrid Classe que define um Grid Uniform (TELEA, 2008). É classe ancestral de StructuredGrid. Define o conjunto de dados para serem renderizados pela classe “Render”, havendo assim separação entre dados (dataset) e a classe que os renderiza e apresenta (SCHROEDER; YAMROM, 1994).

Classe StructuredGrid Classe que define o Grid Estruturado, provendo o suporte para os dados do radar.

[2]UML Unified Modeling Language ou Linguagem de Modelagem Unificada. Conjunto de símbolos e definições estabelecidas por Ivar Jacobson, James Rumbaugh e Grady Booch para modelagem de aplicações

Figura 21: DIAGRAMA DE CLASSES



FONTE: o Autor

Classe GridRender Classe para renderização de um dataset; é a classe ancestral de RadarGridRender.

RadarGridRender Classe responsável pela renderização dos dados do Radar.

Point Classe que define um ponto com atributos de posição, bem como atributo de cor (*color* do ponto.

ListPoints Classe define uma simple lista de pontos, definindo a geometria de cada elemento de informação do dataset. Define um tipo de célula para o grid por meio de seu conjunto de pontos.

Color Classe que representa uma cor.

ColorTable Classe que representa uma tabela de cores para ser utilizada na renderização dos dados.

As próximas seções detalham como essas classes e funções inter-relacionam-se para produzir a apresentação dos dados de radar no navegador.

4.2.1 Dados do Radar

Os dados do radar são armazenados em arquivos contendo os dados em formato binário, compactados em formato específico da empresa Sigmet©, fornecedora do equipamento.

Para a aplicação de renderização, a extração de dados é efetuada pela leitura do arquivo dos dados do radar (que contém os dados brutos de leitura) e a partir desse arquivo, extrai-se o conjunto de elevações. Para cada elevação, extrai-se o conjunto de raios e de cada raio, os conjuntos de *bins*. Em cada *bin*, é extraído o valor da refletividade. A aplicação responsável pela extração dos dados é uma aplicação desenvolvida em Java.

A comparação de Java com outras linguagens Orientadas a Objetos, mostra propriedades bem definidas sem alguns pequenos problemas encontrados em C++, conforme (NAMI, 2008). Quanto ao desempenho, o trabalho de (BULL et al., 2001) mostra que o desempenho em aplicações científicas está muito próxima de aplicações escritas em C ou Fortran.

O formato dos dados de saída é o formato JSON [3] pela praticidade para conversão em objetos JavaScript, bem como pelo pouco uso de metadados (diferentemente de formatos baseados em XML).

A figura 22 mostra o diagrama de classes contendo o relacionamento entre as classes que representam a saída dos dados após a leitura.

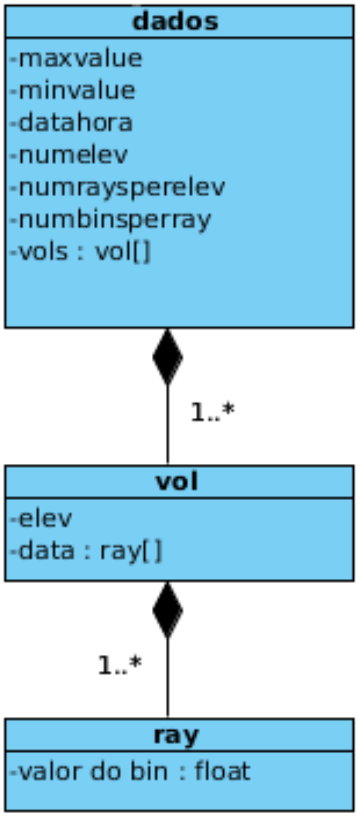
Descrição dos atributos das classes :

Classe dados

- maxvalue : valor máximo da faixa de leitura dos dados;
- minvalue : valor mínimo da faixa de leitura dos dados;
- datahora : data e hora da leitura dos dados em horário GMT;
- numelev : número de elevações contidas no conjunto de dados;

[3] Javascript Object Notation - (INTRODUCING..., 2012)

Figura 22: DIAGRAMA DE CLASSES - DADOS DO RADAR



FONTE: o Autor

- numraysperelev : número de raios por elevação;
- vols : lista contendo os dados de cada elevação

Classe vol

- elev : ângulo de elevação;
- data : lista contendo dados dos bins

Classe ray

- valor de cada *bin* : valor lido do arquivo de dados do radar. Devido à notação JSON, não há necessidade de identificador e o próprio valor pode ser diretamente inserido na lista de valores de cada raio.

Exemplo de dados em formato JSON:

”maxvalue”:72.0,”minvalue”:-24.0,”datahora”:”01/04/2011 19:50”,”numelev”:1,

”numraysperelev”:360,”numbinsperray”:800,”vols”:[”elev”:0.50,”data”:

[[-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-6.0,-4.0, ...], [-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-999.0,-6.0,-4.0 .....]]

Os dados em formato JSON são disponibilizados para aplicação JavaScript que é executada no navegador. A notação JSON permite que os dados sejam carregados e “transformados” em objetos Javascript, sendo então transformados em objeto dados, composto por atributos simples e um atributo matriz (objeto Array) denominado *vols*. Cada elemento da matriz vols é um objeto do tipo vol, que contém um atributo denominado *data*, que é uma matriz de objetos ray, onde cada elemento é o valor do dado de um *bin*.

Para efetuar os testes, foram extraídos os dados de cada elevação e armazenados em arquivos separados. Um arquivo contendo os dados de todas as elevações foi também criado. Desse modo, para apresentação dos dados, deve-se alterar o arquivo a ser carregado.

4.2.2 Criar Grade

Esta etapa é responsável pela formação do *dataset*, ou seja, pelo conjunto entre a grade e os dados associados a esta grade.

A primeira etapa no processo é o estabelecimento da grade, vinculada com a geometria dos dados, que por sua vez é baseada em coordenadas esféricas devido à captura dos dados fornecida pela etapa anterior de aquisição de dados. Os dados estão baseados em :

- Alcance
- Azimute
- Elevação

Alcance é dado pela multiplicação do intervalo entre *bins* e o número do *bin*.

Azimute é o ângulo que corresponde ao raio onde são obtidos os dados. Faixa de valores é 0° a 359°.

Elevação corresponde ao ângulo de elevação da antena de radar em relação à horizontal.

Com esses três valores obtêm-se coordenadas esféricas. O suporte para os dados é feito pela grade estruturada, conforme (TELEA, 2008, p.65). Tem-se então a grade, definindo-se todas as coordenadas para cada *bin*, em modo sequencial, ou seja, a partir do primeiro raio (ou azimute), calculam-se todos os bins desse raio; ao final de um raio, inicia-se o outro e assim sucessivamente.

A estrutura de células é definida estabelecendo a estrutura geométrica sem vínculo com os dados (SCHROEDER; YAMROM, 1994) e (TELEA, 2008, p.89).

Os dados são inseridos em uma estrutura construída pela instância de objetos DataAttribute, com comprimento igual ao comprimento dos dados adquiridos na etapa anterior.

Com a estrutura de células e os dados, forma-se então a estrutura de Dataset, definida com a geometria da Grade e com os dados, sendo então definido o valor na instância da classe StructuredGrid, conforme (OLIVEIRA JR; SCHEER; SATO, 2012, p.9), sendo preenchido no objeto de renderização.

4.2.3 Calcular Matriz

Esta etapa refere-se aos cálculos de perspectiva, rotação e translação necessários para visualização dos dados por meio do modelo de câmera implementado.

A visualização dos dados ou objetos tridimensionais em um tela de computador é similar a câmera fotográfica (digital ou não) que registra cenas tridimensionais, sensibilizando um filme (ou sensor de captação digital) que é bidimensional. Para que exista o efeito de profundidade, distância e perspectiva há necessidade de definição de um modelo de câmera e de cálculos de perspectiva para correta representação.

WebGL não possui definição para câmera ((CANTOR; JONES, 2012, p.105)), logo a implementação para os cálculos de matrizes de visualização e perspectiva precisam ser feitas explicitamente pelo desenvolvedor. Portanto, as operações de visualização e de cálculo de perspectiva precisam ser implementadas para que os objetos possam ser renderizados na tela.

A visualização dos dados na tela é produzida mediante um conjunto de transformações lineares de espaço R^3 do modelo de objeto para o espaço R^2 do monitor ou tela onde serão projetados os elementos. Conforme (GOMES; VELHO, 2008, p.344-345), a definição de um modelo de câmera virtual genérico é um processo que envolve transformações entre sete sistemas de coordenadas, associados a sete espaços : Espaço do Objeto, Espaço de Cena, Espaço de Câmera, Espaço Normalizado (ou de Normalização), Espaço de Ordenação, Espaço de Ima-

gem e Espaço de Dispositivo.

Espaços

Os vários espaços determinam sistemas de coordenadas que têm aplicação específica dentro do pipeline para apresentação dos dados.

Espaço de Objeto é o espaço associado a cada objeto e cujo sistema de coordenadas depende da geometria do objeto.

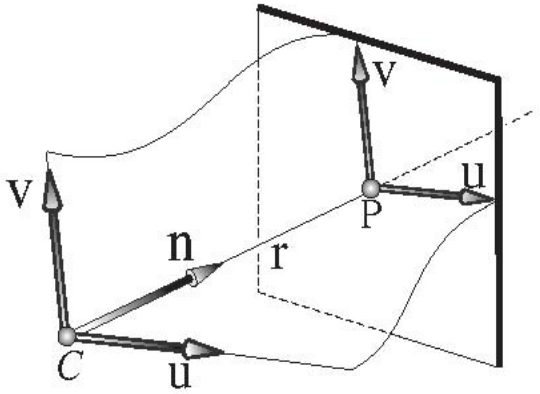
Espaço de Cena ou de Mundo é o sistema de coordenadas global, comum a todos os objetos.

Espaço de Imagem é o espaço definido por um sistema de coordenadas no plano de projeção onde se encontra a tela virtual;

O Espaço de Câmera é o espaço definido pelo sistema de coordenadas associado à projeção cônica, utilizado para definir a **câmera virtual**, ou seja, definir a posição e orientação da câmera em relação ao sistema global. Nesse sistema são definidos alguns parâmetros tais como Tela Virtual, Distância Focal e Centro de projeção.

A posição da câmera é dada pelo Centro de Projeção ou Centro Ótico. Definindo um referencial a partir desse ponto, utilizando três vetores. Um dos vetores define o Eixo Ótico. O Plano de Projeção situa-se à distância d, denominada Distância Focal. Outros dois vetores, $\vec{v}$ e $\vec{u}$ complementam e formam o referencial.

Figura 23: PLANO DE PROJECAO



FONTE: (GOMES; VELHO, 2008, p.347)

A figura 23 mostra o ponto C, os eixos de orientação e o Plano de Projeção.

O Espaço de Imagem é o espaço do Plano de Projeção do Espaço da Câmera. Na figura 23, percebe-se que o eixo ótico (eixo $\vec{n}$) "fura" o Plano de Projeção no ponto P, denominado Ponto Principal, e com os vetores $\vec{u}$ e $\vec{v}$, formam um referencial ortonormal. Define-se então uma "janela" retangular que é denominada Tela Virtual. O Plano de Projeção (ou Plano de Visão (FOLEY et al., 1990, p. 237)) é associado à sigla VPN, e o vetor $\vec{n}$ é denominado Normal do Plano de Visão, utilizando a sigla VPN.

Conforme a figura 24, têm-se:

$Q = (u_{min}, v_{min})$

$S = (u_{max}, v_{max})$

A largura para a Tela Virtual é:

$l_u = u_{max} - u_{min}$

Definindo $s_u = \frac{l_u}{2}$, então:

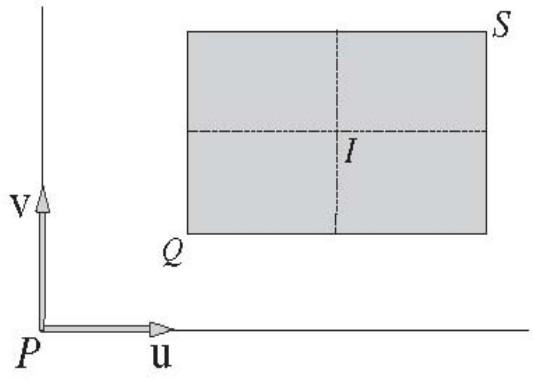
$2s_u = u_{max} - u_{min}$.

De forma análoga, obtém a altura da tela virtual como:

$2s_v = v_{max} - v_{min}$.

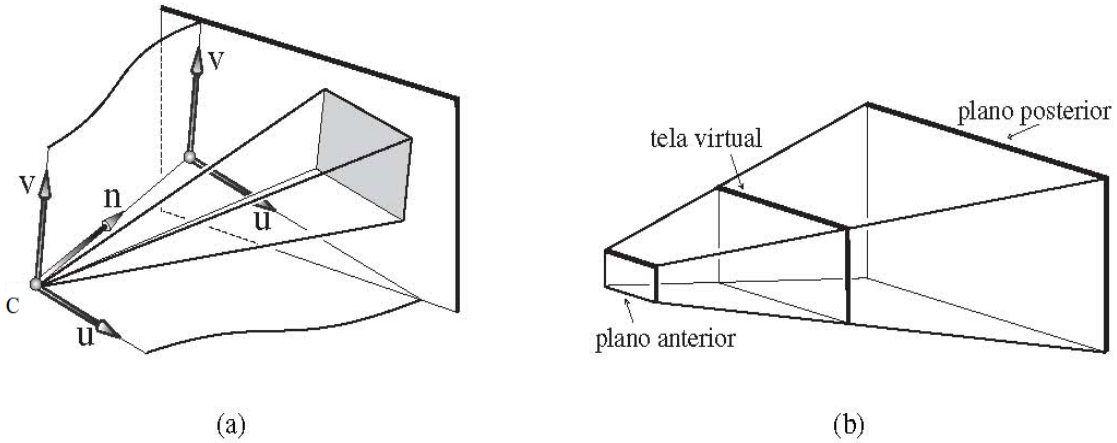
O **Volume de Visão** é definido pela pirâmide formada pelo Centro de Projeção C e pelo Plano de Projeção, mostrado na figura 25. A parte (a) da figura mostra a Pirâmide de Visão. Um modelo de projeção genérico não contém a Pirâmide de Visão alinhada com o Eixo Ótico, ou seja, a altura da Pirâmide não está sobre o Eixo Ótico. Porém, como será visto adiante, neste trabalho adota-se o Eixo Ótico alinhado com o Volume de Visão.

Figura 24: TELA VIRTUAL



FONTE: (GOMES; VELHO, 2008, p.347)

Figura 25: VOLUME DE VISÃO



FONTE: (GOMES; VELHO, 2008, p.348)

São definidos dois planos na Pirâmide de Visualização, o Plano Anterior (ou Plano Próximo) e Plano Posterior (ou Plano Distante). A distância do ponto C (ver figura 23 até o Plano Anterior é denominada *n* e a a distância do ponto C distância ao Plano Posterior é denominada f (GOMES; VELHO, 2008, p.348). A figura (b) de 25 mostra os dois planos, que formam o Tronco de Pirâmide que é o Volume de Visão. Segundo (WATT, 2000, p.148), em termos práticos adota-se que o Plano Próximo coincidente com o Plano de Visão.

A idéia principal do Volume de Visão é a de possibilitar o *Clipping* [4], ou seja, apenas objetos que estão dentro desse volume terão projetados sobre a Tela Virtual e serão visualizados pelo observador.

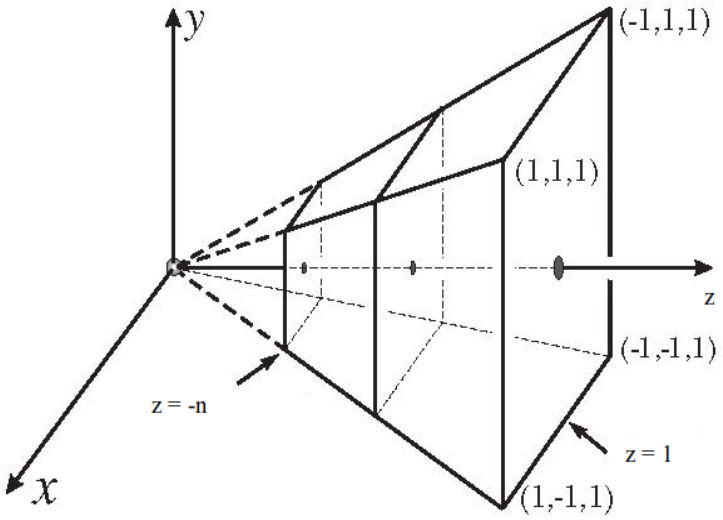
O Espaço Normalizado é o espaço utilizado para definir o *Clipping* dos objetos, selecionando aqueles que serão vistos ou renderizados. É formado pelo Volume de Visão, porém limitado por $-z \leq x \leq z$, $-z \leq y \leq z$ e z variando de um valor mínimo até 1. Associando o vetor $\vec{u}$ ao eixo x, o vetor $\vec{v}$ ao eixo y e o vetor $\vec{n}$ ao eixo z, fica estabelecido um referencial para o Volume de Visão, e pode-se utilizar as coordenadas x, y e z. Esse espaço facilita as operações de recorte, bem como prepara para o Espaço de Ordenação. Os elementos geométricos desse espaço sofrem uma Transformação Projetiva, visando que os elementos sejam projetados sobre a Tela Virtual.

O Espaço de Ordenação é o espaço que possui um sistema de coordenadas que visa facilitar a operação de visibilidade, determinando a posição de cada objeto na cena de modo que possa ser possível a percepção de qual objeto está mais próximo ou distante. Basicamente,

[4]Clipping significa literalmente um pedaço que foi cortado de algo. Em Computação Gráfica representa a operação que permite que apenas objetos que estão dentro do Campo de Visão da câmera sejam mostrados, ou seja, o sistema ou programa que está renderizando os objetos não mostra objetos que não podem ser visualizados.

é o espaço Normalizado que sofre uma "deformação" para que seja transformado em um Cubo, de modo que as operações para determinar a profundidade z dos elementos permitam classificar os objetos, identificando qual elemento está mais próximo da Tela Virtual (GOMES; VELHO, 2008, p.349-350). Esse espaço também é denominado Coordenadas de Projeção Normalizadas [5] ((WATT, 2000, p.159). Os valores permitidos para as coordenadas são valores $-1 \leq x, y, z \leq +1$.

Figura 26: VOLUME DE VISÃO NORMALIZADO



FONTE: (GOMES; VELHO, 2008, p.349), adaptado pelo Autor

O Espaço do Dispositivo é o espaço do dispositivo gráfico que fará a apresentação dos objetos geométricos. Nesse espaço é feita uma transformação de modo que os dados projetados na Tela Virtual, sejam projetados no dispositivo.

Câmera Virtual

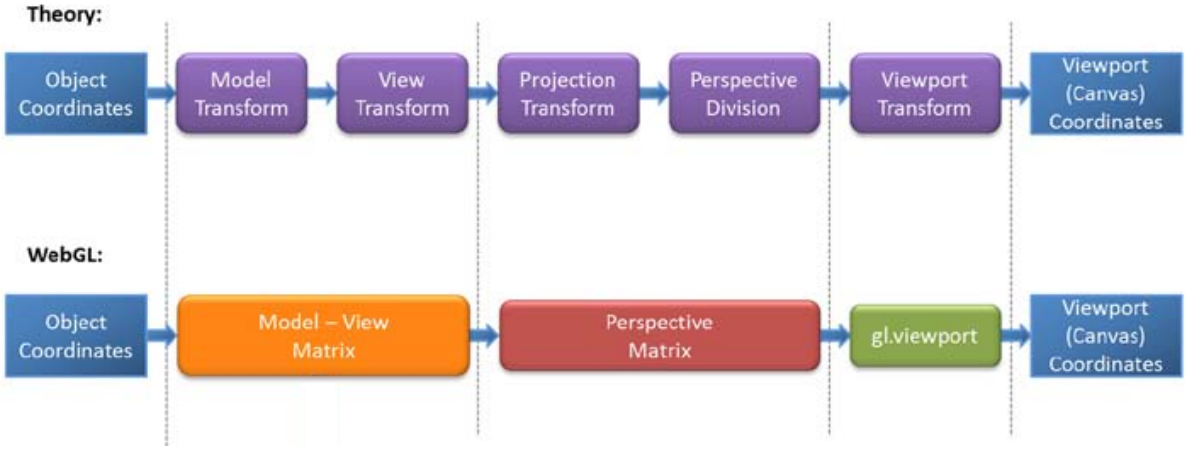
Para obter a projeção dos elementos geométricos no dispositivo, precisam ser calculadas as várias transformações lineares que "convertem" os pontos em cada um dos espaços mencionados anteriormente até o Espaço do Dispositivo, ou seja, fazem a mudança de coordenadas entre os vários sistemas até que a imagem seja renderizada no dispositivo.

A figura 27 mostra dois diagramas mostrando as transformações entre os vários espaços e as matrizes que são necessárias para implementação da câmera em WebGL.

A relação entre as transformações e os espaços propostos por (GOMES; VELHO, 2008) e o visto nos diagramas de (CANTOR; JONES, 2012, p.115) é :

[5] NPC - Normalized Projection Coordinates

Figura 27: TRANSFORMAÇÕES ENTRE ESPAÇOS E IMPLEMENTAÇÃO



FONTE: (CANTOR; JONES, 2012, p.115), adaptado pelo Autor

- *Model Transform* e textitView Transform são transformações entre os espaços de Objeto e de Cena, e desse com o espaço de Câmera
- *Projection Transform* e textitPerspective Division são transformações dos espaços de Câmera, Normalizado, Ordenação e de Imagem
- *Viewport Transform* é a transformação para o espaço de Dispositivo.

A figura 27 destaca que o modelo de câmera utilizado neste trabalho contém apenas duas matrizes que sintetizam as transformações entre os vários espaços, cabendo a última operação no dispositivo diretamente pelo *shader* de vértices aplicando as duas matrizes em cada vértice do objeto :

- Matriz Modelo-Visualização (ou *Model-View matrix*)
- Matriz de Perspectiva (ou *Perspective matrix*)

Matriz Modelo-Visualização A Matriz Modelo Visualização ou *Model-View* é a matriz que efetua a translação do Sistema de Coordenadas da Câmera (ponto C na figura 23) bem como operações de rotação em relação aos eixos x, y e z, caso sejam aplicadas rotações na posição da câmera.

Esta matriz está diretamente vinculada a forma de interação do usuário com o movimento da câmera dentro do Volume de Visualização. Dois modelos muito comuns são a câmera baseada nos ângulos de rotação em relação a origem e a câmera denominada *LookAt*[6].

[6]Look At significa literalmente “olhar para”

O primeiro modelo, a câmera é fixa e apenas rotaciona em relação a seus eixos e baseia-se nas matrizes de translação da posição da câmera e dos ângulos de rotação em relação aos eixos coordenados.

A matriz de translação ao ponto $C = (C_x, C_y, C_z)$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & C_x \\ 0 & 1 & 0 & C_y \\ 0 & 0 & 1 & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Denominando os ângulos rotação em relação ao eixo x como α, eixo y como β e eixo z como γ, tem-se a matrizes de rotação para cada eixo conforme (GOMES; VELHO, 2008, p. 87-88) e (WATT, 2000, p. 5) :

$$\mathbf{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R_y} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

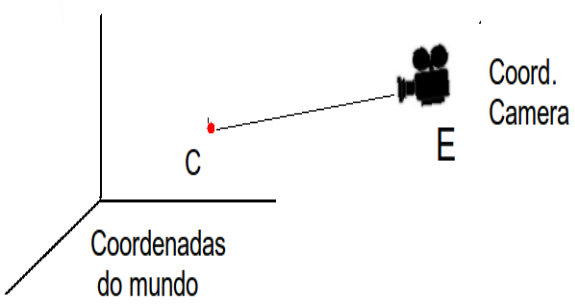
$$\mathbf{R_z} = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A aplicação destas matrizes de rotação em cada eixo, tem-se a matriz resultante:

$$\mathbf{R} = \begin{pmatrix} \cos(\beta)\cos(\gamma) & \cos(\gamma)\cos(\alpha)\sin(\beta) & \cos(\alpha)\cos(\gamma)\sin(\beta)+\sin(\alpha)\sin(\gamma) & 0 \\ \cos(\beta)\sin(\gamma) & \cos(\alpha)\cos(\gamma)+\sin(\alpha)\sin(\beta)\sin(\gamma) & -\cos(\gamma)\sin(\alpha)+\cos(\alpha)\sin(\beta)\sin(\gamma) & 0 \\ -\sin(\beta) & \cos(\beta)\sin(\alpha) & \cos(\alpha)\cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

O modelo de câmera *LookAt* baseia-se na posição da câmera (ponto E também denominado *Eye* [7]), no ponto de interesse (ponto C), ou seja, para onde a câmera está "olhando" e no vetor de referência e que posiciona a câmera para cima. Esta câmera tem a vantagem que permite mudar o ponto (ponto C) que se deseja observar, dando uma "sensação" de direção.

Figura 28: CÂMERA "LOOKAT"



FONTE: (LANDEMAN, 2008, p.3), adaptado pelo Autor

Conforme (HILL JR, 2001, p.358-367) e (LANDEMAN, 2008), esse modelo de câmera é composto por três parâmetros :

- Ponto de interesse C (C_x, C_y, C_z)
- Posição da câmera E (E_x, E_y, E_z)
- Vetor *Up* [8], que indica qual é a posição da câmera, ou qual é a posição que indica a posição voltada para cima.

A partir dos três parâmetros, os outros três vetores são criados para formar a base da câmera: os vetores $\vec{n}$, $\vec{v}$ e $\vec{u}$. Além disto, a origem do sistema passa a ser o ponto E, ou seja, o ponto da posição da câmera (LANDEMAN, 2008, p.11).

- vetor n é o vetor obtido a partir do vetor entre os pontos C e ponto E. Deve-se normalizar o vetor.

$$\vec{n} = \frac{\overrightarrow{CE}}{|\overrightarrow{CE}|}$$

[7] Eye da língua inglesa significa olho
[8] Up é expressão da língua inglesa que significa acima, para cima.

- vetor u é obtido fazendo o produto vetorial entre o vetor n e o vetor Up U;

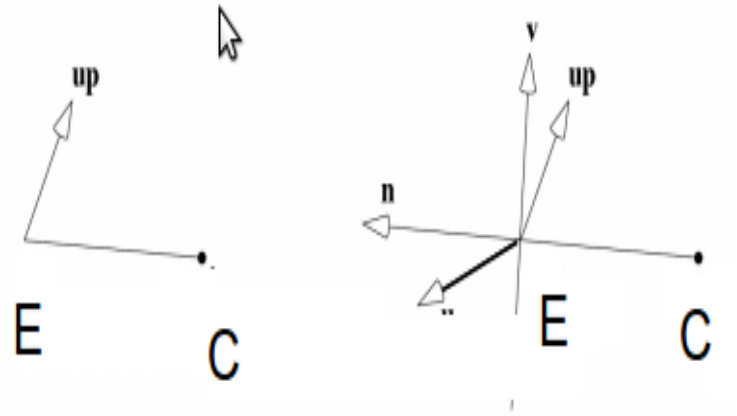
$$\vec{u} = \frac{\overrightarrow{Up} \times \vec{n}}{| \overrightarrow{Up} \times \vec{n} |}$$

- vetor v é obtido pelo produto vetorial entre o vetor n e o vetor u.

$$\vec{v} = \vec{n} \times \vec{u}$$

A figura 29 mostra os vetores da base.

Figura 29: VETORES $\vec{n}$,$\vec{u}$ e $\vec{v}$



FONTE: (LANDEMAN, 2008, p.12), adaptado pelo Autor

Deseja-se obter a matriz de Modelo Visualização para o modelo de câmera. Para isto, conforme (LANDEMAN, 2008) e (HILL JR, 2001), esta matriz corresponde a matriz formada pela rotação do sistema de coordenadas da câmera e pela translação do sistema de coordenadas desta câmera.

A matriz de rotação é formada pelas componentes dos versores da base:

$$\mathbf{R} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A matriz de translação da origem é dada pela matriz:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A matriz resultante M é o produto da matriz de translação pela matriz de rotação, ou seja, primeiramente **translada-se** e depois **rotaciona-se**.

$$\mathbf{M} = \begin{pmatrix} u_x & u_y & u_z & -\vec{e}\cdot\vec{u} \\ v_x & v_y & v_z & -\vec{e}\cdot\vec{v} \\ n_x & n_y & n_z & -\vec{e}\cdot\vec{n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta é a matriz de Modelo-Visualização necessária para a transformação de visualização da câmera. Além dos pontos de interesse (ponto C) e da posição da câmera (ponto E), ainda pode-se utilizar os ângulos de *Yaw*, *Roll* e *Pitch* [9]. *Yaw* é o ângulo que estabelece a direção da câmera e é a rotação em relação ao eixo u. *Roll* é a giro em relação ao angulo longitudinal da câmera, ou seja, eixo dado pelo vetor $\vec{n}$. *Pitch* é a inclinação da câmera em relação ao eixo v.

Pela flexibilidade que o modelo de câmera proporciona para que o usuário possa interagir com o ambiente e a câmera, foi implementado esse modelo. A classe Camera contém os atributos que correspondem aos vetor $\vec{n}$,$\vec{u}$ e $\vec{v}$. Logo, a matriz Modelo-Visualização é a matriz M da câmera *LookAt*.

Esse modelo permite que se "navegue" pelo ambiente virtual, "passeando" com a câmera.

Matriz de Perspectiva A matriz de perspectiva é a matriz que efetua as transformações entre os espaços de modo que o usuário tenha a sensação da perspectiva e profundidade da imagem que está sendo projetada na tela.

Esta matriz efetua, como já mencionado, a transformação entre os Espaços de Câmera, Normalizado, de Ordenação e de Imagem. Antes das definições da matriz, faz-se necessário efetuar algumas definições de parâmetros da Câmera Virtual para estabelecer o modelo geométrico do Espaço de Câmera.

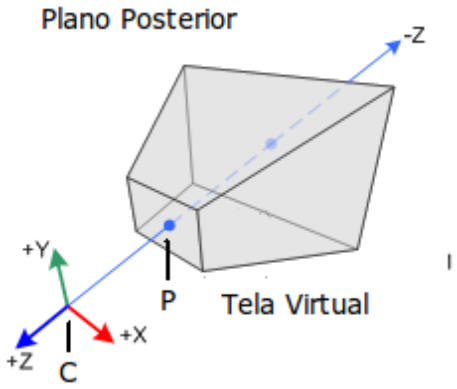
[9] Esta nomenclatura para esses ângulos são denominações emprestadas de termos aeronáuticos e francamente utilizadas na denominação na literatura em Computação Gráfica, inclusive literatura em língua portuguesa. Dado isto, a nomenclatura foi mantida.

A matriz de perspectiva depende do modelo de Câmera selecionado. Para implementação, selecionou-se tipo de Câmera Câmera de Perspectiva [10]. Dentro desse tipo de câmera, optou-se pelo modelo de câmera definido pelo OpenGL, que possui as seguintes características :

- Eixo ótico coincidente com o Eixo de Visão
- Ângulo de Visão determina a distância focal (distância até o plano próximo)
- Todo ponto dentro do Volume de Visão é projetado no Plano de Visão, que está no Plano Próximo
- O Sistema de Coordenadas Normalizadas encontra-se no centro do cubo normalizado que possui dimensões $-1 \leq x, y, z \leq +1$.

O Eixo Ótico coincidente com o Eixo de Visão facilita as transformações, visto que o Ponto Principal P coincide com o centro da Tela Virtual (ponto I na figura 24). A figura 30 mostra a configuração geométrica e posição do ponto P, inclusive com eixo de coordenadas e ponto C.

Figura 30: EIXO ÓTICO



FONTE: (HO, 2012b), adaptado pelo Autor

O Ângulo de Visão é o parâmetro que permite identificar a distância focal, a partir de simples cálculos trigonométricos, e que definem outros parâmetros da câmera que são a distância focal, largura e altura da Tela Virtual. A figura 31 mostra a altura da tela virtual. Pode-se obter o valor pela relação ((GOMES; VELHO, 2008, p.351)):

[10] Outros modelos ou tipos de câmera existentes: câmera afim, câmera de perspectiva frace (weak perspective), câmera ortográfica ((GOMES; VELHO, 2008, p.343)

$$\alpha = 2arctg\frac{PA}{CP} \Longleftrightarrow tg(\frac{\alpha}{2}) = \frac{PA}{CP} \tag{4.1}$$

Onde : CP é a distância focal, que é a distância até o Plano Próximo; PA é a altura da Tela Virtual s_v

Observação : A Tela Virtual é definida como tendo formato quadrado. Apenas na transformação para o Espaço do Dispositivo, é que a relação de aspecto [11] é aplicada e são obtidas os valores de largura e altura da Tela Virtual a partir das informações físicas do dispositivo gráfico onde a imagem será apresentada.

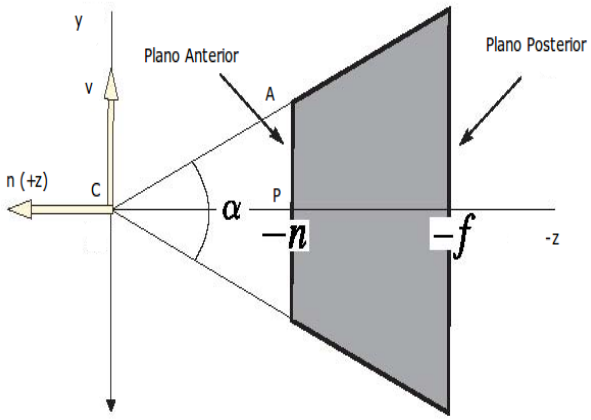
Então

$$tg(\frac{\alpha}{2}) = \frac{s_u}{-n} \tag{4.2}$$

E

$$tg(\frac{\alpha}{2}) = \frac{s_v}{-n} \tag{4.3}$$

Figura 31: ÂNGULO DE VISÃO

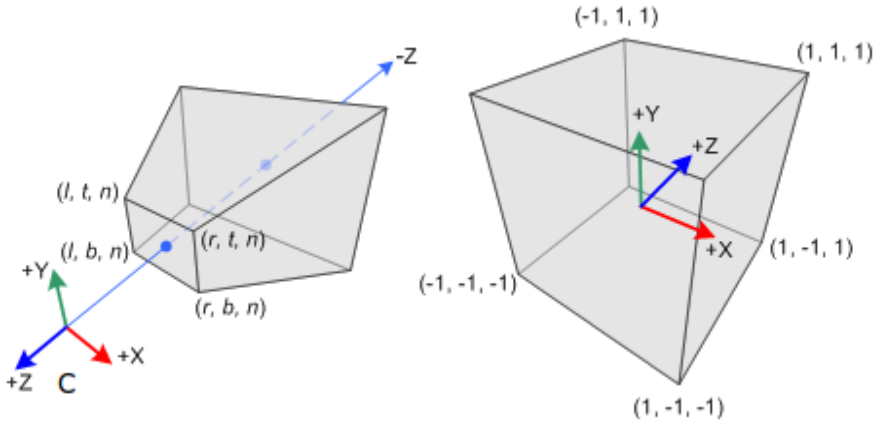


FONTE: (GOMES; VELHO, 2008, p.367), adaptado pelo Autor

A figura 32 mostra os dois volumes: Volume de Visão e o Cubo com coordenadas normalizadas. Observa-se que o sistema de coordenadas normalizadas tem o eixo z apontando em direção contrário ao eixo z do sistema de coordenadas do Volume de Visão.

[11] Relação entre largura total da tela com a sua altura

Figura 32: COORDENADAS NORMALIZADAS

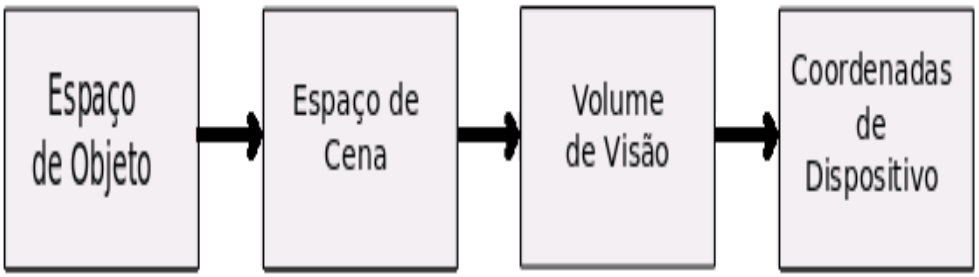


FONTE: (HO, 2012b), adaptado pelo Autor

Transformação da Câmera

Com a escolha do modelo similar ao utilizado pelo OpenGL, pode-se simplicar algumas transformações, pois estas transformações já contemplam transformações em mais de um espaço. A figura 33 mostra os vários espaços e sistemas de coordenadas onde serão aplicadas e calculadas as transformações.

Figura 33: PIPELINE DE ESPAÇOS DE TRANSFORMAÇÃO



FONTE: o Autor

A Transformação da Câmera é uma transformação projetiva que é o conjunto das várias transformações entre os vários espaços, a partir do Espaço de Cena (ou Global ou de Mundo). A Transformação do Espaço de Objeto para o Espaço de Cena é uma Transformação não-projetiva. Então faz-se necessário obter cada uma das transformações entre os espaços que efetuam a projeção do objeto na Tela Virtual.

Busca-se então uma matriz de transformação de modo que transforme pontos no sistema de coordenadas de mundo no R^3 para o espaço R^3 do Espaço de Tela:

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \\ w_{ndc} \end{pmatrix} = M_{proj} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

Onde

- x_{ndc}, y_{ndc} e z_{ndc} são as coordenadas do ponto no espaço normalizado do dispositivo
- a matriz M_{proj} é a matriz de projeção que transforma os pontos no Espaço de Cena no Espaço do Dispositivo.

Observam-se os termos w_{ndc} e w_e, que são termos devido a notação de Coordenadas Homogêneas [12]

Esta matriz é a matriz formada pelas várias matrizes de cada transformação entre cada espaço.

Transformação para Espaço Normalizado e de Ordenação

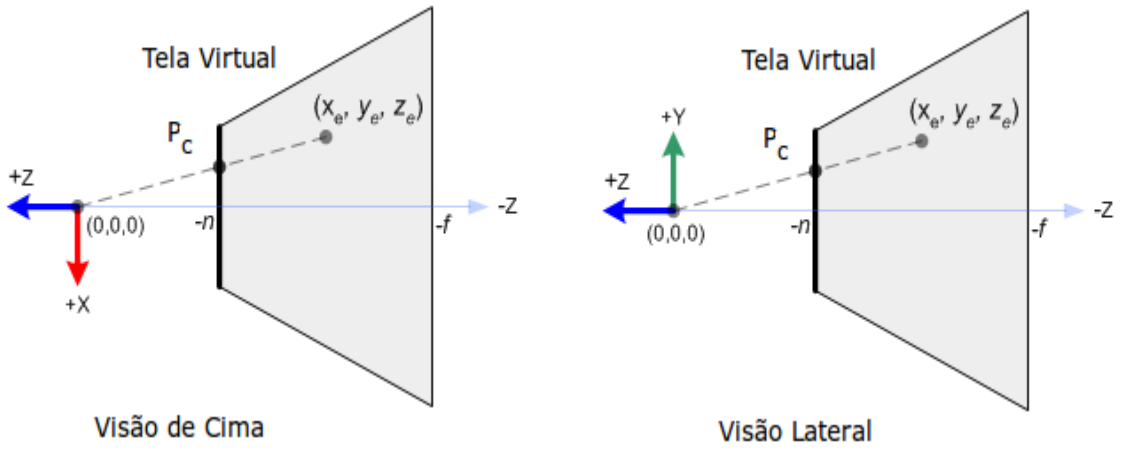
Precisa-se obter uma transformação (ou matriz de transformação) entre o Espaço de Cena e os Espaços Normalizado e de Ordenação. Busca-se portanto uma matriz 4 x 4, denominada S. Chamando um ponto qualquer $P_c = (x_c, y_c, z_c)$, esse é dado por:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \mathbf{S} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

A figura 34 mostra esquematicamente a projeção de ponto $P_e = (x_e, y_e, z_e)$ na Tela Virtual. O Ponto $P_c = (x_c, y_c, z_c)$ na Tela Virtual é calculado a partir da semelhança de triângulos, formados pelas coordenadas dos pontos.

[12] Coordenadas Homogêneas foram introduzidas por August Ferdinand Möbius para facilitar o cálculo de perspectivas, representando um Espaço de dimensão N com N+1 coordenadas. Desta forma, um ponto (X,Y) em coordenadas cartesianas é representado por (x,y,w) em coordenadas Homogêneas. Para converter novamente para Coordenadas Cartesianas, aplicam-se : $X = x/w$ e $Y = y/w$. Com estas coordenadas é possível ter um ponto "movido" para o infinito dado por (x,y,0). (GOMES; VELHO, 2008, p.41) (HO, 2012a)

Figura 34: PROJEÇÃO DE PONTOS NA TELA VIRTUAL



FONTE: (HO, 2012b), adaptado pelo Autor

Para a coordenada x_c

$$\frac{x_c}{-n} = \frac{x_e}{-z_e} \tag{4.4}$$

$$x_c = \frac{x_e - n}{-z_e} \tag{4.5}$$

Para normalizar, efetua-se a divisão pelo metade da largura da tela virtual s_u. Tem-se :

$$x_c = \frac{x_e - n}{-z_e s_u} \tag{4.6}$$

Da equação 4.2, a equação 4.6, tem-se:

$$x_c = \frac{x_e}{-z_e tg(\frac{\alpha}{2})} \tag{4.7}$$

Para a coordenada y_c e utilizando 4.3, tem-se:

$$y_c = \frac{y_e}{-z_e tg(\frac{\alpha}{2})} \tag{4.8}$$

Percebe-se que as duas equações tem um divisor comum: z_e. Como devemos expressar nossa matriz em coordenadas homogêneas, tem-se :

$$w_c = -z_e \tag{4.9}$$

Então, reescrevendo x_c e y_c, tem-se:

$$x_c = \frac{x_e}{tg(\frac{\alpha}{2})} \tag{4.10}$$

$$y_c = \frac{y_e}{tg(\frac{\alpha}{2})} \tag{4.11}$$

Para o termo z_c, a dedução é um pouco mais complexa pois deve-se permitir que visualmente, a sensação de profundidade seja mantida após aplicada a normalização.

Supondo uma relação linear entre z_c e z_e, pode-se expressar:

$$z_c = Az_e + B \tag{4.12}$$

Sabe-se que um dos parâmetros da câmera é justamente a normalização do espaço de Ordenação em um cubo com dimensões $-1 \leq x,y,z \leq +1$. Para normalizar, divide-se por $-z_e$.

$$z_c = \frac{Az_e + B}{-z_e} \tag{4.13}$$

Então, conhece-se 2 valores para z_e na normalização: Quando $-z_e = -n$, $z_c = -1$ e quando $-z_e = -f$, $z_c = +1$. Logo:

$$-1 = \frac{Az_e + B}{-n} \tag{4.14}$$

$$+1 = \frac{Az_e + B}{-f} \tag{4.15}$$

Obtem-se:

$$A = -\frac{f+n}{f-n} \tag{4.16}$$

$$B = -\frac{2fn}{f-n} \tag{4.17}$$

Então:

$$z_c = -\frac{f+n}{f-n}z_e - \frac{2fn}{f-n} \tag{4.18}$$

A matriz S é dada por :

$$\mathbf{S} = \begin{pmatrix} \frac{1}{tg(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tg(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

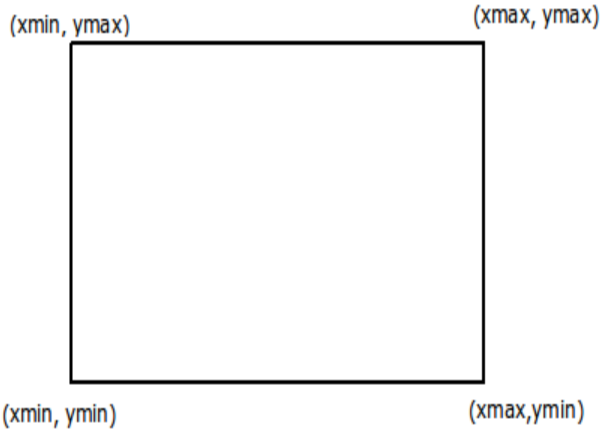
Transformação para Espaço de Tela

A transformação para o Espaço de Tela deve levar em conta que o Espaço de Tela é regido pelas coordenadas físicas do Dispositivo. Busca-se portanto uma matriz 4 x 4, denominada D que efetua a tranformação do Espaço de Ordenação para o Espaço do Dispositivo.

Chamando um ponto qualquer na Tela do Dispositivo de $P_d = (x_d, y_d, z_d)$, esse é dado por:

$$\begin{pmatrix} x_d \\ y_d \\ z_d \\ w_d \end{pmatrix} = \mathbf{D^{-1}} \begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix}$$

Figura 35: COORDENADAS DE DISPOSITIVO



FONTE: o Autor

Na figura 35, temos as dimensões da tela são x_{min}, x_{max}, y_{min} e y_{max}. Do Espaço Normalizado, temos que as coordenadas da Tela Virtual, são definidas pelos pontos (-1,-1), (+1, -1), (+1,+1) e (-1,+1). Como a Tela Virtual é menor que a Tela Real do Dispositivo, precisa-se estabelecer uma relação linear de escalonamento para as coordenadas x e y.

Tem-se para a coordenada x_d:

$$x_d = Ax_c + B \tag{4.19}$$

Mapeando, tem-se:

$$x_{max} = A \cdot 1 + B \quad x_{min} = A \cdot (-1) + B \tag{4.20}$$

Conduzindo a :

$$A = \frac{x_{max} - x_{min}}{2} \quad B = \frac{x_{max} + x_{min}}{2} \tag{4.21}$$

Aplicando na equação 4.19, temos:

$$x_d = \left(\frac{x_{max} - x_{min}}{2} \right) x_c + \frac{x_{max} + x_{min}}{2} \tag{4.22}$$

Aplicando o mesmo para a coordenada y_d, temos:

$$y_d = \left(\frac{y_{max} - y_{min}}{2} \right) y_c + \frac{y_{max} + y_{min}}{2} \tag{4.23}$$

O dispositivo ou tela não possui nada relacionado com a coordenada z.

Então a matriz $\mathbf{D^{-1}}$ é composta por:

$$\mathbf{D^{-1}} = \begin{pmatrix} \frac{x_{max}-x_{min}}{2} & 0 & 0 & \frac{x_{max}+x_{min}}{2} \\ 0 & \left(\frac{y_{max}-y_{min}}{2}\right) & 0 & \frac{y_{max}+y_{min}}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Porém toda a dedução foi feita mapeando-se o ponto da Tela Virtual para a tela do dispositivo, mas precisa-se da Transformação do dispositivo para a tela Virtual.

Deve-se portanto, obter a matriz Inversa de $\mathbf{D^{-1}}$

$$\mathbf{D} = (\mathbf{D^{-1}})^{-1} = \begin{pmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & \frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & \frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A matriz de transformação projetiva M_{proj} é então definida como ((LENGYEL, 2004, p.127)) :

$$\mathbf{M_{proj}} = \mathbf{D} \cdot \mathbf{S} \tag{4.24}$$

$$\mathbf{M_{proj}} = \begin{pmatrix} -\frac{2n}{(x_{max}-x_{min})s_u} & 0 & \frac{x_{max}+x_{min}}{x_{max}-x_{min}} & 0 \\ 0 & -\frac{2n}{(x_{max}-x_{min})s_v} & \frac{y_{max}+y_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Implementação

A implementação da câmera (matrizes e transformações) é feita na classe Camera. Também a classe WebGL define atributos e métodos responsáveis pela visualização dos objetos dentro do contexto WebGL. A instância de Camera é utilizada pela classe WebGL por meio do atributo camera desta. A figura 36 mostra as duas classes.

Na classe WebGL, os atributos que definem os parâmetros da câmera são :

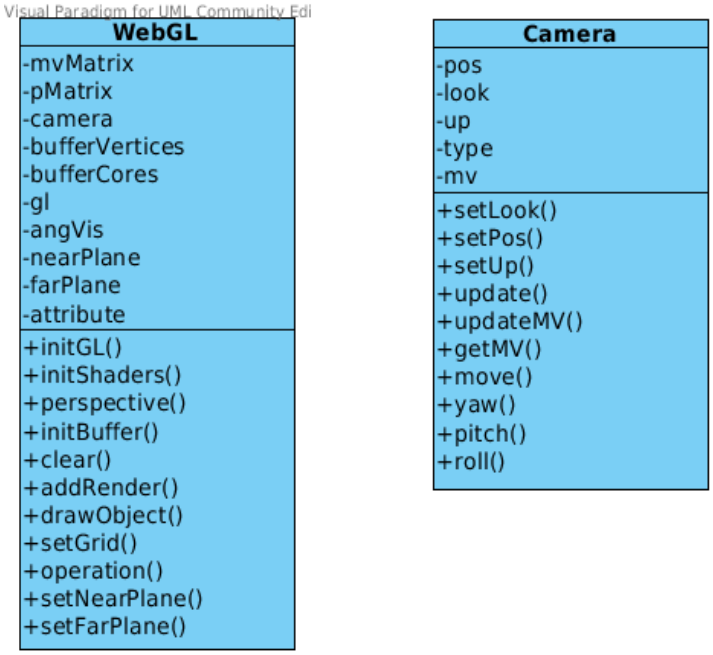
- nearPlane : plano Próximo do volume de visualização;
- farPlane : plano Distante do volume de visualização;
- angVis : ângulo de Visão.

Esses valores podem ser alterados pelos métodos setNearPlane, setFarPlane e setAngVis, mas no momento da inicialização do objeto WebGL pelo seu constructor [13], o plano Próximo e o plano Distante são setados em 1 e 10000 respectivamente, enquanto o valor do Ângulo de Visão é iniciado em 45°.

A classe WebGL representa o contexto para renderização de objetos no navegador. Para efetuar a renderização, o método drawObject, recebendo como parâmetro a instância do

[13]Constructor é o método especial responsável pela criação da instância da classe. Pode ou não inicializar atributos

Figura 36: CLASSE WEBGL E CLASSE CAMERA



FONTE: o Autor

objeto RadarGridRender. Esse método é o método-chave no cálculo da matriz, pois invoca o método perspective definido em WebGL, calculando a matriz de perspectiva da câmera e inserindo no atributo pmatrix de WebGL. O cálculo da matriz de perspectiva é efetuado pela função utilitária *makeperspective*, que recebe 4 parâmetros : Ângulo de Visão, razão de aspecto, posição do plano próximo e posição do plano distante.

O método *makeperspective* utiliza o ângulo de visão e a razão de aspecto para calcular os valores de x_{max}, x_{min}, y_{max}, y_{min} utilizando as equações 4.3 para o cálculo de y_{max}. O valor de y_{min} é definido com $-y_{max}$, pois na equação 4.3, s_v é definido como sendo metade da altura da tela. A partir dos valores de y_{max} e y_{min}, obtem-se x_{max} multiplicando y_{max} pela Relação de Aspecto e x_{min} multiplicando y_{min} pela Relação de Aspecto.

O cálculo da matriz de Perspectiva (4.2.3) é feito por meio da function makeFrustum, que é invocada logo após os cálculos de y_{max}, y_{min}, x_{max} e x_{min}, sendo seu resultado inserido no atributo pmatrix.

Após o cálculo da matriz de perspectiva, é invocado o método *draw* do próprio objeto RadarGridRender. Nesse método é efetuado o cálculo da translação e rotação em relação aos eixos, sendo o resultado inserido no atributo mvMatrix.

Os valores dos atributos pMatrix e mvMatrix são setados nos uniforms do *shader* de

vértices pelo método *setUniforms* da classe WebGL. Esse método primeiramente obtém os valores das matrizes na forma de uma matriz de valores de ponto flutuante (*float*)e depois invoca *uniformMatrix4fv* do objeto do contexto WebGL. Esse método vincula a matriz de float com a variável *uniform* definida no *shader* de vértice e *shader* de fragmento.

O trecho de código abaixo mostra as operações para ambas as matrizes e seus *uniforms*:

```
f = new Float32Array(this.mvMatrix.getValuesByColumns());
this.gl.uniformMatrix4fv(this.shaderProgram.mvMatrixUniform, false, f);
f32 = new Float32Array(this.pMatrix.getValuesByColumns());
this.gl.uniformMatrix4fv(this.shaderProgram.pMatrixUniform, false, f32);
```

O uso de *uniforms* disponibiliza as matrizes para o *shader* de vértices como um valor constante para todas as instância do *shader* de vértices. Enquanto que para cada vértice, WebGL cria e executa o código do *shader* de vértice em paralelo com outros vértices. Os valores dos atributos de vértices são obtidos na próxima etapa - Extrair Vértices.

A classe Camera define métodos resposáveis pelo posicionamento da câmera, bem como de cálculos da matriz de Modelo-Visualização :

- setLook : seta o atributo look, que define o ponto de interesse (ponto C);
- setPos : seta o atributo pos, que é a posição da câmera (ponto E)
- setUp : seta o atributo up que é o vetor Up.
- move : move a câmera no eixo do vetor $\vec{n}$, ou seja, no eixo do ponto de interesse (C) e da posição da câmera (E).
- yaw : efetua rotação do ângulo de Yaw, calculando rotação em relação aos eixos $\vec{n}$ e $\vec{u}$ em relação ao eixo $\vec{v}$;
- roll : efetua rotação do ângulo de Roll, calculando rotação em relação aos eixos $\vec{u}$ e $\vec{v}$ em relação ao eixo $\vec{n}$;
- pitch : efetua rotação do ângulo de Pitch, calculando rotação em relação aos eixos $\vec{n}$ e $\vec{v}$ em relação ao eixo $\vec{u}$;
- update é o método que recalcula os vetores $\vec{n}$, $\vec{u}$ e $\vec{v}$. Esse método deve ser invocado após alteração de qualquer dos principais atributos *Look*, *Pos* e *Up*. É invocado internamente.
- updateMV é o método que atualiza a matriz Modelo-Visualização.

A classe WebGL deve receber a instância da classe Camera no seu atributo *camera* de modo que possa ser utilizada no momento de "desenhar" os objetos.

A última etapa da figura 27 é efetuada pelo código executado pelo *shader* de vértice, por meio da obtenção da posição do vértice obtido pela multiplicação das matrizes de Modelo-Visualização e de Perspectiva pela posição do vértice. O trecho de codigo do *shader* de vértice abaixo mostra esta etapa:

```
gl\_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
```

Onde

gl_position é variável pré-definida pelo WebGL é corresponde a posição de um vértice;

uPMatrix é o nome da variável *uniform* para a matriz de Perspectiva;

uMVMatrix é o nome da variável *uniform* para a matriz de Modelo-Visualização obtida do atributo camera de WebGL

aVertexPosition é o nome da variável *attribute* para o vértice que está sendo calculado.

vec4 é tipo de dado do WebGL para vetores com 4 dimensões, normalmente para multiplicação de coordenadas afim.

Deve-se observar que há um valor 1.0 setado na última posição de *vec4*. Isto é necessário para efetuar a correta multiplicação pelas matrizes.

Com isto, finaliza-se a etapa de multiplicação de matriz para efeitos de perspectiva do objeto a ser apresentado, sendo o objeto correspondente aos dados do radar.

4.2.4 Extrair Vértices

Esta etapa prepara os *Vertex Buffer Objects (VBOs)* para os vértices definidos no dataset. Os pontos são extraídos das coordenadas da Grade e setados nos VBOs, e desta forma, renderizado. Para isto, as seguintes operações são definidas:

- Criação de variável tipo *attribute* no *shader* de vértices;
- Criação de VBO;
- Preenchimento do VBO durante a execução

Criação de attribute

A definição da variável do tipo *attribute* é efetuada no código do*shader* de vértices. Para os vértices, a variável denominada *aVertexPosition* é criada com o trecho de código:

```
attribute vec3 aVertexPosition;
```

A variável é definida com sendo do tipo *vec3*, que significa que é uma variável vetor, contendo três dimensões, *x*, *y* e *z*.

Durante a execução do código, o contexto WebGL recebe o valor de cada um dos vértices a partir do *buffer* VBO definido no código Javascript.

Criação do VBO

A criação do *buffer* (VBO) é efetuada durante a execução do método initbuffers da classe WebGL . O método é invocado pelo método initGl.

A criação de um *buffer* VBO é muito simples, pois é simplesmente por meio do método createBuffer do contexto WebGL. O trecho de código abaixo mostra como foi implementado:

```
this.bufferVertices = this.gl.createBuffer();
```

Onde :

- this é uma referência ao próprio objeto, que nesse caso é o objeto da classe WebGL;
- this.bufferVertices é o nome do VBO, que é um atributo da classe WebGL;
- this.gl é o atributo do contexto WebGL da classe WebGL
- createBuffer é a chamada ao método de criação do *buffer*.

Preenchimento do VBO

Uma vez criado, o *buffer* deve ser preenchido com os vértices durante a execução do código. Isto é efetuado a partir da matriz contendo os vértices utilizando o método bindBuffer do contexto WebGL.

Esta extração de vértices e preenchimento do VBO é efetuada dentro do método draw logo após o cálculo das matriz de posicionamento e perspectiva da câmera (ver 4.2.3), utilizando a função utilitária loadBuffersFromObj.

A função loadBuffersFromObj recebe 4 parâmetros:

- variável do contexto WebGL
- a instância do objeto a ser renderizado
- variável de referência do *shader* de vértices
- Buffer de Vértices
- Buffer de Cores

Onde:

A variável do contexto WebGL é o atributo *context* da instância do objeto WebGL.

A instância do objeto é a instância de RadarGridRender que é responsável por conter os atributos do objeto a ser renderizado, que é o objeto contendo os dados (dataset), bem como atributos de posição do objeto. Os vértices são definidos pelos dados contidos no *dataset*.

O *buffer* de Vértices é o *buffer* WebGL que conterá os valores dos vértices definido anteriormente. Corresponde ao atributo bufferVertices do objeto da classe WebGL.

Para efetuar a extração dos vértices do objeto são seguidos os passos abaixo :

- Invoca-se bindBuffer do contexto WebGL;
- Os vértices são obtidos do objeto (parâmetro obj);
- Invoca-se o método bufferdata do contexto WebGL;
- Invoca-se o método vertexattribpointer.

BindBuffer é método do contexto WebGL (conforme (KhronosGroup Inc, 2011)) que recebe 2 parâmetros: o tipo de dados (array (matriz) ou elemento) e a referência ao *buffer*. No código desenvolvido, o tipo de dados é o `ARRAY_BUFFER`, indicando que os dados devem ser tratados com matriz. A referência ao *buffer* é o parâmetro bufferVertices.

Os vértices são extraídos e inseridos em uma variável temporária mbuffer em forma de matriz pelo uso do método getVertexAsArray do objeto , no caso, a instância de RadarGridRender. A variável é então preenchida com a sequência de todos os vértices do objeto, onde cada elemento da matriz é uma coordenada do vértice.

O método bufferdata do contexto WebGL preenche o *buffer* de vértices. Similar ao que acontece com códigos de OpenGL, bufferdata insere os valores no *buffer* previamente definido por meio da função bindBuffer, no caso bufferVertices. São ainda definidos dois atributos de bufferVertices: itemSize e numItems. Itemsize é o tamanho de cada item da matriz, que no caso é três devido as três dimensões (ou coordenadas) que define cada um dos vértices. NumItems é o número de itens da matriz, que é o comprimento da matriz dividido por três.

O método vertexattribpointer é o método do contexto WebGL que preenche (ou prepara para preenchimento durante a execução do *shader* de vértices) segundo ((CANTOR; JONES, 2012, p.33)). uma variável do *shader* de vértices. No código tratado nesta etapa, a variável *attribute* é aVertexPosition definida no *shader* de vértices com o *buffer* corrente (definido por bindBuffer). O método recebe 4 parâmetros: a variável associada à variável attribute do *shader* de vértices, o comprimento do *buffer*, o tipo de dado de cada posição do *buffer*, o tamanho do passo e o "*offset*" ou ponto de início do *buffer*.

A variável associada a variável *attribute* do *shader* de vértices é definida pelo método initShaders, invocado na inicialização do ambiente WebGL. O trecho de código abaixo demonstra como é efetuado :

```
this.shaderProgram.vertexPositionAttribute =
       this.gl.getAttribLocation(this.shaderProgram,"aVertexPosition");
```

Percebe-se que a variável do tipo attribute "aVertexPosition" do *shader* de vértices está associada com shaderProgram.vertexPositionAttribute.

O trecho de código abaixo da função utilitária loadBuffersFromObj mostra as operações efetuadas:

```
    webGL.bindBuffer(webGL.ARRAY_BUFFER, bufferVertices);
    var mbuffer = obj.getVertexAsArray();
    webGL.bufferData(webGL.ARRAY_BUFFER, new Float32Array(mbuffer),
webGL.STATIC_DRAW);
    bufferVertices.itemSize = 3;
    bufferVertices.numItems = mbuffer.length / 3;
    webGL.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    bufferVertices.itemSize, webGL.FLOAT, false, 0, 0);
```

Com este conjunto de operações, os vértices são extraídos do objeto, transformados em VBO e passados ao *shader* de vértices.

4.2.5 Aplicar Tabela de Cores

Esta etapa aplica a tabela de cores, definida no momento da inicialização, a cada um dos vértices do dataset do objeto RadarGridRender.

A tabela de cores é definida na function criarRadarGridRender, que inicializa a origem e também define a tabela de cores pelo método setColorTable definido na classe RadarGridRender.

Similar a etapa anterior, quando os vértices foram setados no *buffer* (VBO), a cor de cada vértice deve ser setado em um *buffer* (VBO).

A própria função utilitária loadBuffersFromObj contém as chamadas similares a etapa anterior, porém a matriz temporária é preenchida com os dados de cores, onde, para cada vértice, tem-se o valor para red (vermelho), green (verde) e blue (azul), que compõem a cor de um vértice.

Criação de attribute

A definição da variável do tipo *attribute* é efetuada no código do *shader* de vértices e é denominada *aVertexColor*, criada com o trecho de código:

```
attribute vec4 aVertexColor;
```

Criação do VBO

A criação do *buffer* (VBO) é efetuada durante a execução do método initbuffers da classe WebGL . O método é invocado pelo método initGl.

A criação de um *buffer* VBO é muito simples, pois é simplesmente pelo método createBuffer do contexto WebGL. O trecho de código abaixo mostra como foi implementado:

```
this.bufferCores = this.gl.createBuffer();
```

Onde :

- this é uma referência ao próprio objeto, que neste caso é o objeto da classe WebGL;
- this.bufferCores é o nome do atributo da classe WebGL que conterá as cores de todos os vértices;

- this.gl é o atributo do contexto WebGL da classe WebGL
- createBuffer é a chamada ao método de criação do *buffer*.

Preenchimento do VBO

Após a extração dos vértices (conforme a etapa anterior), a função loadBuffersFromObj preenche o *buffer* e associa à variável do *shader* de vértices.

O trecho de código mostrado a seguir demonstra como é efetuado:

```
  webGL.bindBuffer(webGL.ARRAY_BUFFER, bufferCores);
  mbuffer = obj.getColorAsArray();
  webGL.bufferData(webGL.ARRAY_BUFFER, new Float32Array(mbuffer),
webGL.STATIC_DRAW);
  bufferCores.itemSize = 4;
  bufferCores.numItems = mbuffer.length / 4;
  webGL.vertexAttribPointer(shaderProgram.vertexColorAttribute,
    bufferCores.itemSize, webGL.FLOAT, false, 0, 0);
```

Percebe-se que o código é muito similar ao código mostrado da seção anterior, porém agora a variável temporária mbuffer é preenchida com a matriz obtida por meio de getColorAsArray. Também nota-se que o *buffer* de cores (bufferCores) tem comprimento de cada item (itemsize) igual a quatro, pois as cores são compostas de quatro atributos: *Red* (vermelho), *Green* (verde), *Blue* (azul) e *Alpha* , que é a intensidade.

4.2.6 Render

A apresentação da imagem dentro do contexto WebGL é a parte final do método *draw* da instância de RadarGridRender.

Após as etapas do cálculo da matriz de Perspectiva e de Modelo Visualização e após os *buffers* serem preenchidos com os vértices e cores, deve-se invocar o método *drawArrays* do contexto WebGL, conforme o trecho de código abaixo:

```
context.gl.drawArrays(context.gl.POINTS, 0,
                      context.bufferVertices.numItems);
```

Onde:

context.gl é o atributo que corresponde ao contexto WebGL ;

drawArrays é o método responsável pela renderização. Invoca os *shader* de vértices e de fragmento no pipeline do WebGL ((KhronosGroup Inc, 2011))

context.gl.POINTS é um valor constante que define que serão renderizados pontos

0 é o primeiro elemento a ser renderizado

context.bufferVertices.numItems é o comprimento ou número de vértices a serem renderizados.

Invocando o método, várias instâncias do *shader* de vértice são criadas e gerenciadas pelo WebGL. O mesmo ocorre para o *shader* de fragmento, renderizando a imagem dos dados do radar na página html da aplicação.

4.2.7 Codificação da Aplicação

Descreveu-se até o momento todo o pipeline do Visualizador, descrevendo-se o conjunto de classes e funções denominadas utilitárias e que foram encapsuladas na biblioteca **utilgl.js**. Também foram descritos os *shaders* que efetuam a renderização dos vértices e fragmentos. Também descreveu-se o leitor de dados de radar, que é responsável pela extração dos dados do radar, entregando em formato JSON de modo que sirva de modelo para renderização pelo contexto WebGL utilizando o conjunto de classe e funções utilitárias.

A aplicação do Visualizador foi desenvolvida utilizado o conjunto de classes e funções utilitárias e também dos *shader*s dentro do arquivo html denominado visualizador.html.

A função *start()* é a função responsável por iniciar ambiente, inicializando o modelo de dados, o contexto WebGL (pela classe WebGL) e os objetos para representação do dataset do radar. Esta função é invocada tão logo a página HTML seja apresentada pelo evento *onload* da tag *body*.

Abaixo, código da function *start()*:

```
function start(){
      camera = new Camera(0,0,0,0,0,0, 1,0,0);
      resetCamera(false);

      criarDadosFromJson();
      criarRadarGridDataset2();
      criarRadarGridRender(new V3(px - 1000 ,py - 2300, pz +0 ));
      ....
```

```
        w= new WebGl();

        var cv = document.getElementById("canvasApp");
        w.initGl("canvasApp", backgroundcolor);
        ..

        w.setBackgroudColor(backgroundcolor) ; ///Color.COLOR_BLACK);

        mostrarRadarGrid();
    }
```

Os passos para apresentação dos dados do radar :

- inicializar o objeto de camera;
- obter os dados via JSON e criar o *dataset*;
- invocar a função criarRadarGridDataset2 para criação do objeto da classe RadarGridRender, que é responsável pela renderização dos dados de radar;
- criar o objeto da classe WebGL, inicializando por meio do método initGl e setando a cor de fundo com preto;
- chamando a função mostrarRadarGrid para mostrar os dados

Outra função importante é a função updateScene(), que é a função responsável pela atualização dos dados na tela.

```
function updateScene() {
    w.camera = camera;
    w.clear();
    mostrarRadarGridRender();
}

 function drawCurrentObj() {
     w.drawObject(currentObj);
 }
```

Acima, tem-se código extraído das duas funções: updateScene e drawCurrentObj.

A função updateScene é a função que efetua a renderização, onde a instância de câmera é passada no atributo camera de WebGL.

A função drawCurrentObj desenha o RadarGridRender na tela.

4.2.8 Exemplo de visualização

Para inicializar o visualizador, basta carregar a página HTML por um servidor web tal como Apache [14] ou Light HTTP [15], ou qualquer outro servidor de páginas HTML.

O uso de um servidor é necessário por razões de segurança do WebGl que não permitem acesso diretamente. Logo, mesmo que a aplicação em código HTML seja executada na mesma máquina, deve-se ter um servidor de aplicações ou de páginas disponibilizando a página.

A opção selecionada para os testes foi o Apache Tomcat, que é um servidor de páginas JSP [16] mantido pela Fundação Apache.

Logo, para acessar a página, deve-se utilizar a URL que contém o contexto da aplicação, no caso **http://localhost:8080/testeswebgl2/visualizador.html**.

A figura 37 mostra um exemplo de visualização da primeira elevação, ângulo de $0,5°$, mostrando toda a tela do visualizador.

Na parte superior, tem-se a imagem com visualização inicial, apresentando os dados de radar da reflectividade 2.1.2. Ao lado direito tem-se a tabela de cores utilizada, variando de -24 a +72, onde -24 é o menor valor, indicando pequenos hidrometeoros e +72 é o maior valor, indicando tempestade extrema.

A figura 38 mostra a seção da imagem e da tabela de cores adotada.

A figura 39 mostra a seção central da página do visualizador.

Nesta seção, tem-se as coordenadas da Câmera em X, Y e Z e também do ponto LookAt que é o ponto de interesse *C* da câmera. O usuário tem o botão "Alterar Câmera", que permite que o usuário possa manualmente inserir valores de coordenadas. Além do botão, tem-se outros dois botões: "Posição inicial" e "Visão Satélite" que auxiliam na forma de visualização.

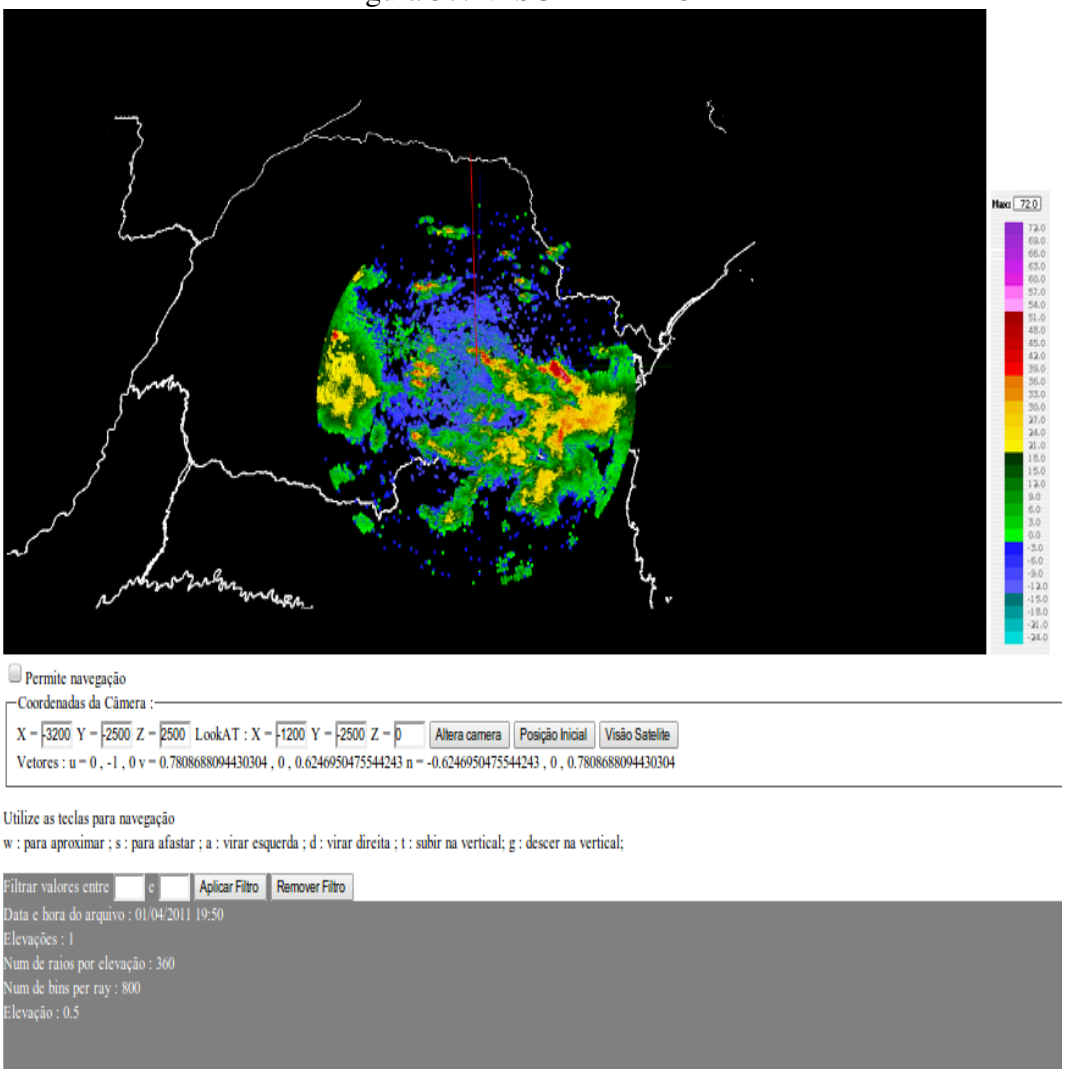
O botão "Posição inicial" reposiciona a câmera em visão inclinada em relação ao radar,

[14] mantido pela Fundação Apache de projetos código aberto. Disponível em http://httpd.apache.org/

[15] Projeto de servidor web código aberto. Disponível em http://www.lighttpd.net/

[16] JSP Java Server Pages. Tecnologia que permite a escrita de páginas utilizando código HTML e também código que permite apresentar conteúdo dinâmico. Necessitam de servidor de aplicações para serem "transformadas" em código HTML puro e serem apresentadas em navegador de Internet

Figura 37: VISUALIZADOR



FONTE: O Autor

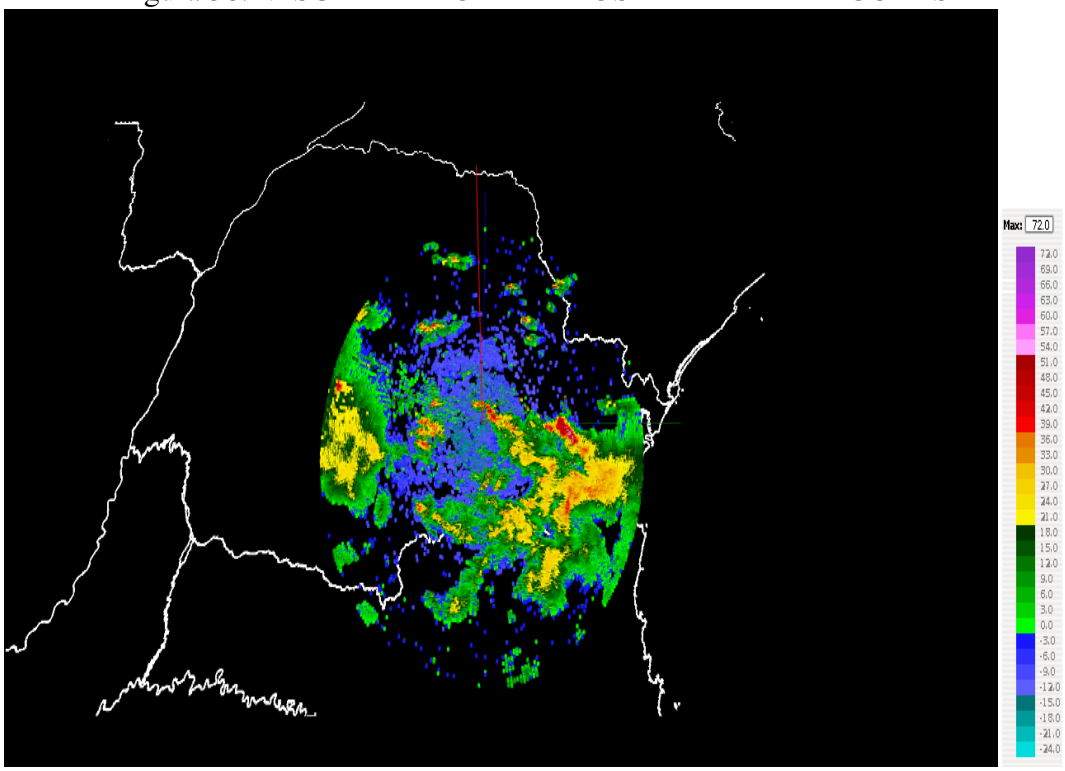
no ponto em que é iniciado o visualizador, enquanto o botão “Visão Satélite”, posiciona a câmera em visão perpendicular, posicionada em coordenada distante do visualizador, simulando uma câmera ou visão a partir de “Satélite fictício”.

No lado esquerdo desta seção, tem-se a caixa de checagem “Permite navegação”, que habilita ou desabilita a navegação pelo usuário. O mecanismo para permitir ou não a navegação deve-se para poupar recursos, visto que na situação de navegação, a aplicação gasta muitos recursos computacionais devido ao reposicionamento da câmera e recálculo de matrizes.

Para navegação utilizando o modelo de câmera implementado, o usuário tem o recurso de utilizar teclas:

- tecla W para deslocamento para frente no eixo da câmera

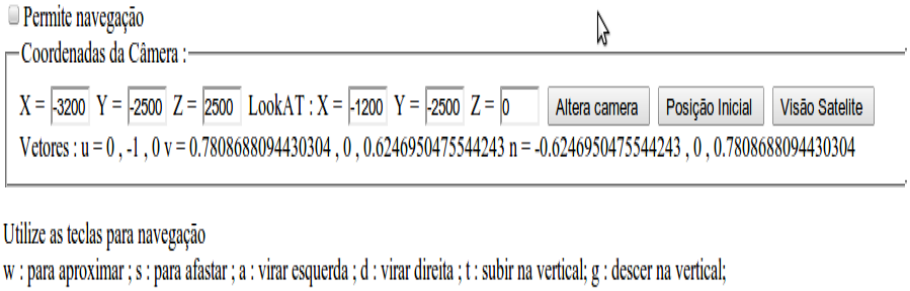
Figura 38: VISUALIZADOR - DADOS E TABELA DE CORES



FONTE: O Autor

- tecla S para deslocamento para trás no eixo da câmera
- tecla D para virar (Yaw) a direita
- tecla A para virar (Yaw) a esquerda
- tecla G para descer a câmera na vertical
- tecla T para subir a câmera na vertical

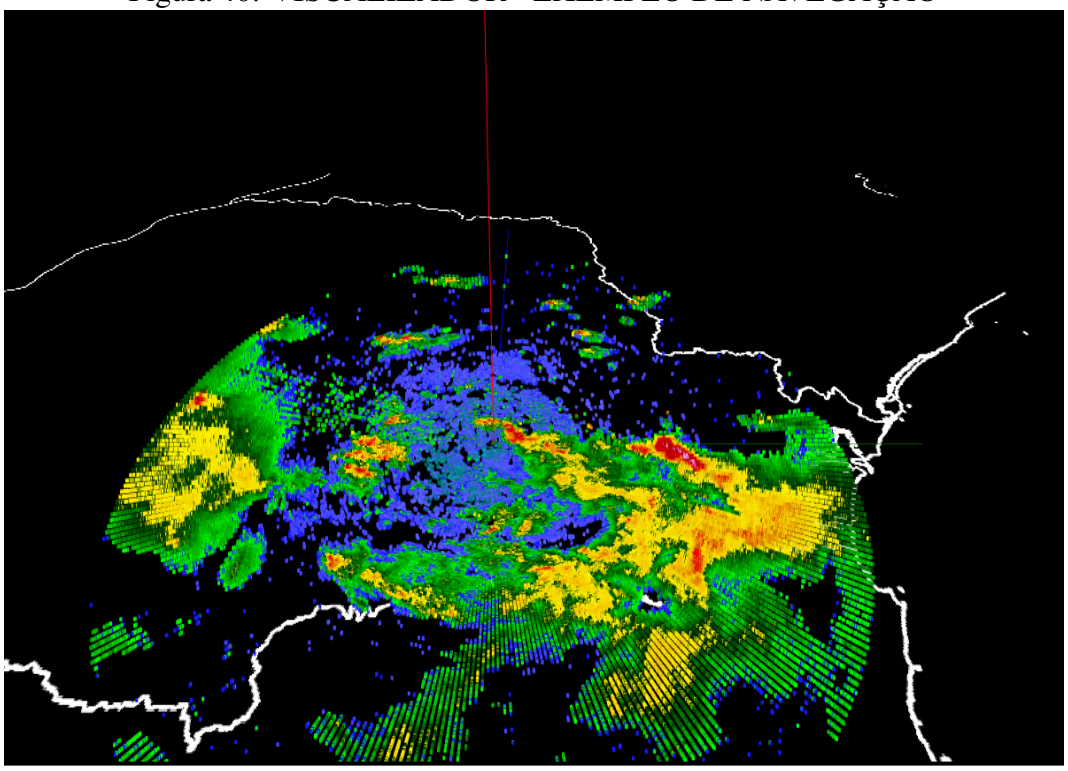
Figura 39: VISUALIZADOR - CONTROLES DE CÂMERA



FONTE: O Autor

Conforme a câmera tem sua posição alterada, as suas coordenadas podem ser vistas nos atributos X, Y e Z, além do ponto em que se está olhando (ponto *C*). Um botão para alterar a posição permite que o usuário entre com coordenadas X, Y e Z tanto para posição da câmera, quanto para o ponto que se deseja observar. As figuras 40 e 41 mostram exemplos de como o usuário pode navegar pelo volume dos dados.

Figura 40: VISUALIZADOR - EXEMPLO DE NAVEGAÇÃO



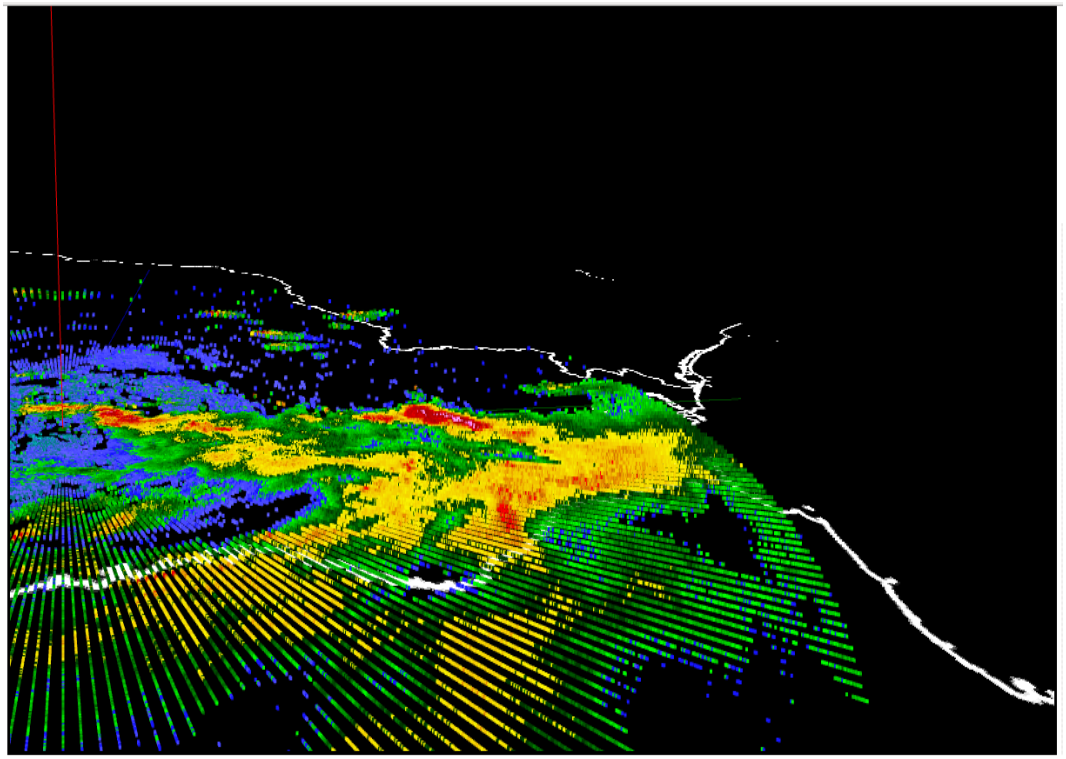
FONTE: O Autor

A parte inferior da tela do visualizador contém o recurso de filtro e também os dados da elevação. A figura 42 mostra os dados da elevação. Os dados informados são :

- Data e hora do arquivo [17]
- Elevações - número de elevações contidas nos dados informados ao visualizador
- Número de raios por elevação
- Número de bin por raio
- Elevação ou Ângulo da Elevação em relação a horizontal

[17] data e hora dos dados coletados pelo radar

Figura 41: VISUALIZADOR - EXEMPLO DE NAVEGAÇÃO



FONTE: O Autor

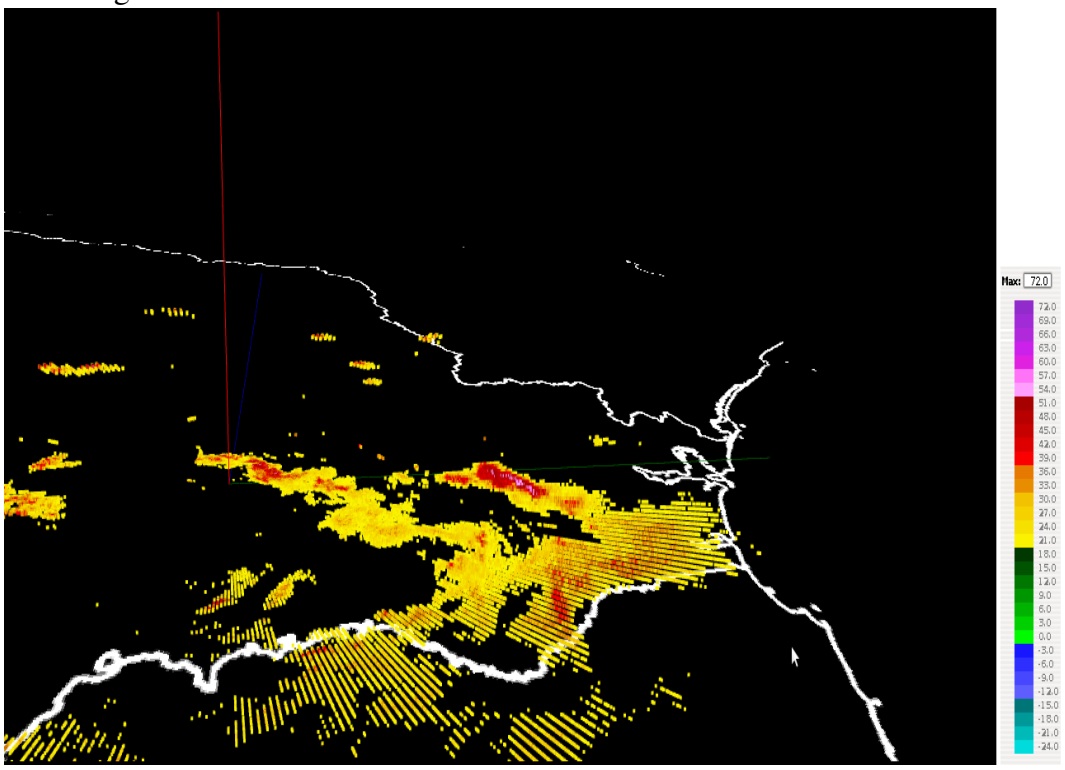
Além dos dados, nesta seção também está disponibilizado o recurso de filtro, pelos campos que permitem ao usuário entrar com os valores a serem filtrados. Pode-se entrar com valores em faixa. Ao acionar o botão “Aplicar Filtro”, os dados são filtrados e a imagem é atualizada. Ao acionar o botão “Remover filtro”, o filtro é removido e todo o conjunto de dados é apresentado.

A figura 43 mostra mostra um exemplo de aplicação do recurso de filtro, onde foram filtrados os dados de reflectividade entre -24 a +21. Percebe-se que o recurso é útil para identificação de dados dentro do conjunto de dados.

Figura 42: VISUALIZADOR - RECURSO DE FILTRO DE DADOS

FONTE: O Autor

Figura 43: VISUALIZADOR - RECURSO DE FILTRO DE DADOS



FONTE: O Autor

4.3 CONSIDERAÇÕES FINAIS

No capítulo, os métodos e técnicas utilizados para o desenvolvimento da aplicação foram apresentados.

O uso da linguagem de Programação Java para o desenvolvimento do leitor de dados justifica-se pela vantagem de compilação única para qualquer plataforma. Além disto, o desempenho comparado com outras linguagens já não é um fator que a distancie de outras linguagens conforme (BULL et al., 2001). Desta forma, esta linguagem é utilizada para codificação da leitura, descompactação de dados e extração dos dados em formato JSON.

O criação de biblioteca com classes e funções em Javascript foi necessário dado que a aplicação será executada em ambiente de Navegador na Internet, bem como o contexto WebGL só pode ser acessado por meio do uso de Javascript. E a notação JSON utilizada também facilita em muito a transferência de dados, permitindo que objetos Javascript sejam imediatamente criados após o recebimento dos dados.

O uso de técnicas de separação de responsabilidades (dados e renderização) segundo (SCHROEDER; YAMROM, 1994) e (TELEA, 2008) estabeleceu a direção para o desenvolvimento das classes dos dados de radar e da classe para renderizar estes dados.

Diferentemente de outros ambientes de programação de visualização tais como OpenGL, WebGL não implementa o modelo de câmera sendo necessária programação e desenvolvimento específicos. O modelo selecionado neste trabalho é o modelo *LookAt*, que permite ao usuário "navegar" dentro do ambiente virtual. O cálculo matricial para cálculos da perspectiva da câmera mostram-se um desafio quando da apresentação dos dados, visto que vários cálculos são necessários, porém quando tratados pelo *shader* de vértice, as matrizes são tratadas como constantes pelos valores *uniforms* para todos os vértices. Porém, todos os vértices são renderizados pelas instâncias do *shader* de vértices, e como apresentado, os valores do *attribute aVertexPosition* é passados um a um para o *shader* de vértice.

Percebe-se também que o desenvolvimento para ambiente "web" ou seja, para navegadores da Internet é diferente do tradicional por se tratar de um ambiente distribuído e também heterogêneo. Associado a este desafio, a linguagem Javascript não possui compilador, sendo os erros de sintaxe descobertos apenas durante a execução.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos utilizando a implementação do visualizador. Por meio da visualização de várias elevações, são apresentados os dados de radar, grandeza refletividade para uma data específica. Além da visualização individual, também será analisada a apresentação de dados de todos as elevações, ou seja, de um volume completo para a data e hora escolhidas.

5.1 CONJUNTO DE DADOS

Para testes com o visualizador, foi selecionada uma data que contivesse grande quantidade de chuva para permitir que as células de chuvas convectivas fossem visualizadas, bem como grandes variações na escala de refletividade Z fossem observadas (valores altos e baixos). A data selecionada foi 01/04/2011, horário 19h50 (7:50PM) em horário GMT [1]. Nesta data e horário houve chuvas de alta intensidade, incluindo granizo em boa parte do estado do Paraná e também em Curitiba 45.

A figura 45 foi obtida em 01/04/2011, por volta das 16h30, horário de Curitiba, que corresponde a 7:30PM GMT. A foto mostra a intensidade do granizo que caiu sobre a cidade de Curitiba nessa data. O jornal local Gazeta do Povo (TRISOTTO; GERON; ANIBAL, 2011), em sua edição do dia 02/04/2011, publicou matéria sobre o evento que trouxe grande transtorno para a cidade, causando problemas ao trânsito e deixando 28 bairros sem energia elétrica devido aos problemas causados pelo vento intenso e pelas descargas elétricas.

5.1.1 Preparação para testes

Os dados coletados pelo radar ficam armazenados em sistema de arquivos do próprio radar, em formato proprietário, como já explicado na seção 4.2.1.

[1] GMT Greenwich Mean Time - horário do meridiano de Greenwich. Em dados meteorológicos são muitas vezes obtidos segundo o horário GMT para não haver problemas com fusos horários

Figura 44: CHUVA INTENSA

FONTE: O Autor

O arquivo do radar contém dados de 13 varreduras com os seguintes ângulos de elevação: $0,5°$, $1,0°$, $1,5°$, $2,0°$, $3,0°$, $4.0°$, $5,0°$, $6,5°$, $8,0°$, $10,0°$, $12,0°$, $15,0°$, $18,0°$ e $21,0°$.

Utilizando-se o leitor de dados de radar, foram gerados arquivos individuais para cada elevação, totalizando quatorze arquivos para cada uma das elevações e também um arquivo contendo todas as quatorze elevações, que compõem o volume completo de dados obtidos ou lidos pelo radar meteorológico.

Para que o visualizador pudesse mostrar os dados, para cada arquivo de dados, a linha de inclusão do arquivo era alterada [2]. Então a linha de número 17 era alterada conforme :

```
<script src="./js/dados_sweep_0.txt"> </script>
```

onde o texto sweep_0.txt deve ser trocado para cada arquivo de dados. Além disso, o servidor foi iniciado como parte do ambiente necessário para execução do visualizador.

Todos os testes foram realizados em computador Notebook marca Lenovo ©, modelo

[2]Nesta versão não foi adotada nenhuma forma de troca de arquivos ou leitura de dados de forma automática e dinâmica, pois a principal observação da aplicação é desempenho para renderização dos dados

Figura 45: FOTOGRAFIA DO JORNAL GAZETA DO POVO

FONTE: (TRISOTTO; GERON; ANIBAL, 2011)

G460, com 4G Bytes de memória RAM, disco rígido de 500 G Bytes e placa de vídeo NVIDIA ©GForce©
310M com 512 Mbytes de memória dedicada ao processamento de vídeo. O sistema operacional Linux Ubuntu ©versão 10.10 64 bits. O driver de vídeo específico fornecido pela NVIDIA©, versão 295.59. Servidor Tomcat [3] versão 7.0. A versão do JDK utilizada para compilar o leitor de dados de radar e para executar o servidor Tomcat é 1.7.0_05-b05. O navegador utilizado foi o Google© Chrome© versão Versão 21.0.1180.89.

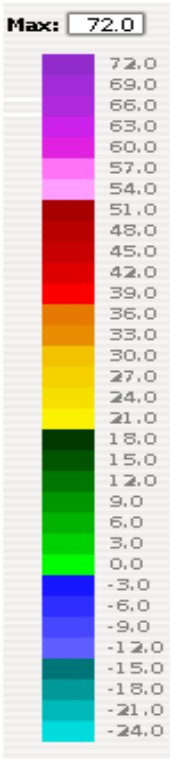
5.1.2 Resultados

5.1.3 Tabela de Cores

Sendo a Refletividade uma grandeza escalar, adota-se tabela de cores para representação dos dados (TELEA, 2008, p.52). A figura 46 apresenta esta tabela de cores e a faixa valores de dBZ correspondente a cada cor.

[3] Servidor de código aberto para tecnologia Java Servlets e Java Server Pages. Disponível em tomcat.apache.org

Figura 46: TABELA DE CORES APLICADA NOS RESULTADOS



FONTE: O Autor

Elevação de $0,5°$

A figura 47 mostra a visualização dos dados relativos à elevação de $0,5°$.

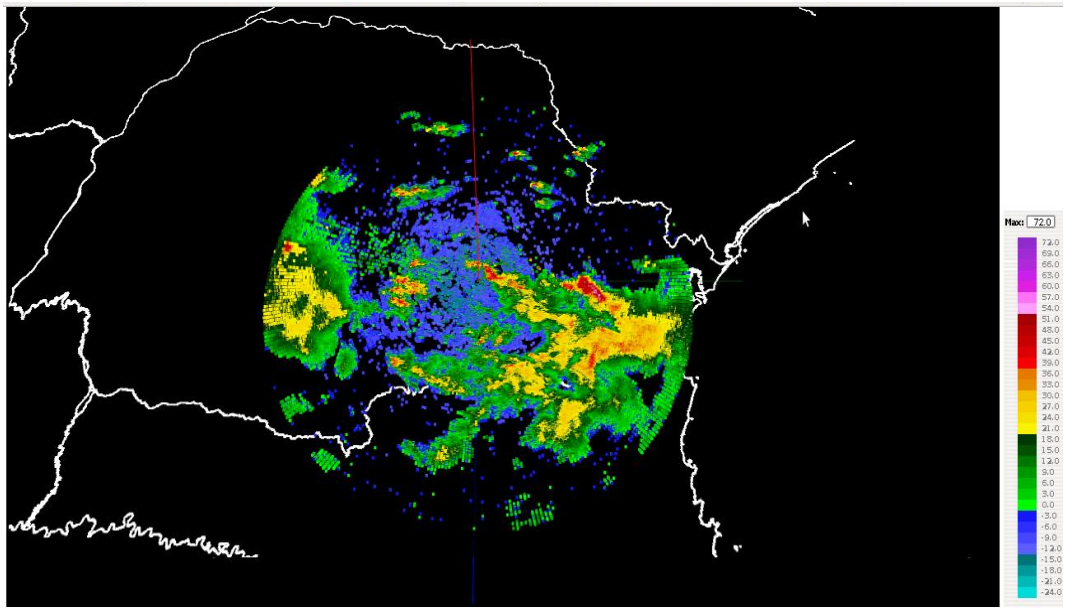
Ativando a navegação (por meio da caixa de checagem “Permite Navegação” 4.2.7) pode-se navegar pelos dados, conforme mostram as figuras 48, 49 e 50.

Ativando o botão “Visão Satélite” e habilitando a navegação dos dados (marcando a caixa de checagem “Permite navegação”), quando navegando diretamente para baixo, observou-se em alguns momentos uma pequena interrupção na apresentação dos dados. Não foi possível determinar com exatidão o tempo da interrupção.

As figuras 51 e 52 mostram a visualização a partir do acionamento do botão “Visão Satelite” e movendo a câmera.

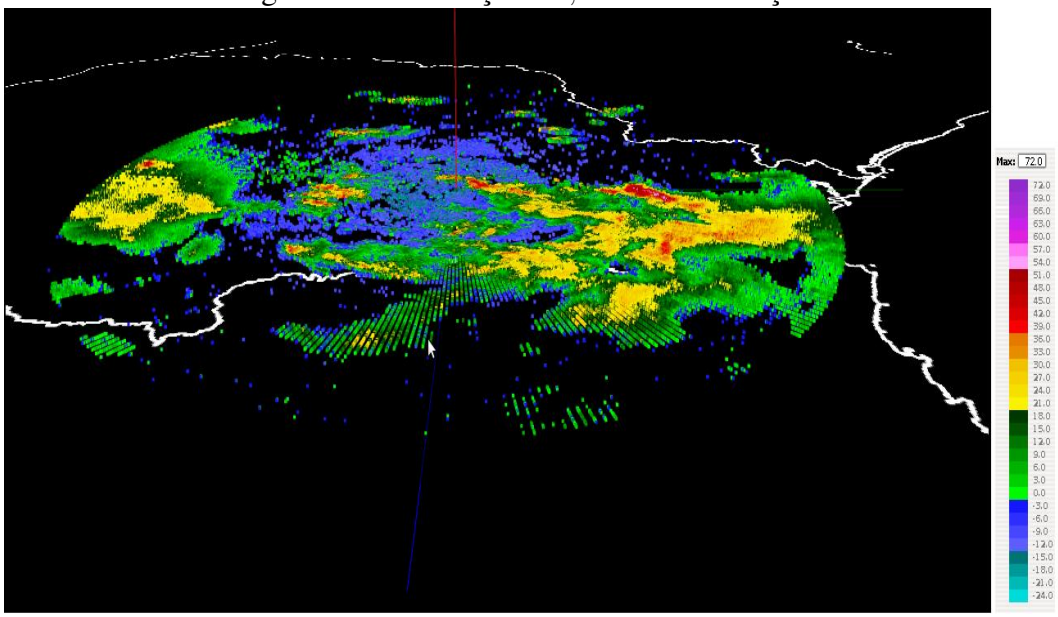
A visualização por meio da “Visão de Satélite” permite comparação com o software em uso atualmente pelo Simepar para visualização dos dados de radar meteorológico, que é ferramenta de visualização que apresenta os dados de maneira planar ou bidimensional. A figura 53 mostra como os dados são representados em software em uso, podendo ser comparado com a figura 54, que foi produzida pela ferramenta de Visualização Tridimensional desenvolvida neste

Figura 47: ELEVAÇÃO $0,5°$



FONTE: O Autor

Figura 48: ELEVAÇÃO $0,5°$ - NAVEGAÇÃO



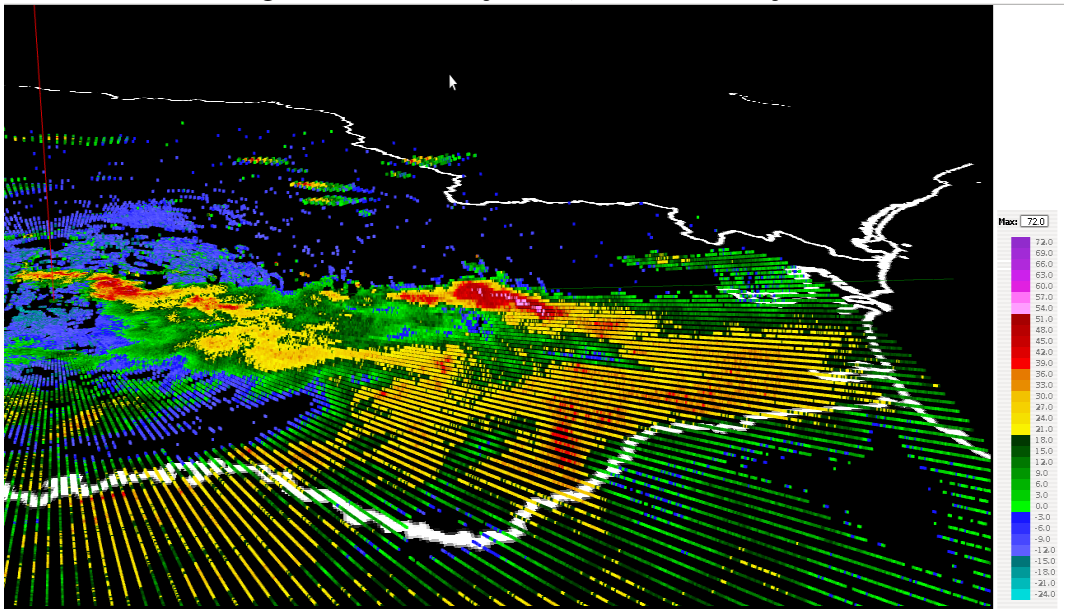
FONTE: O Autor

trabalho.

Elevação de $1,0°$

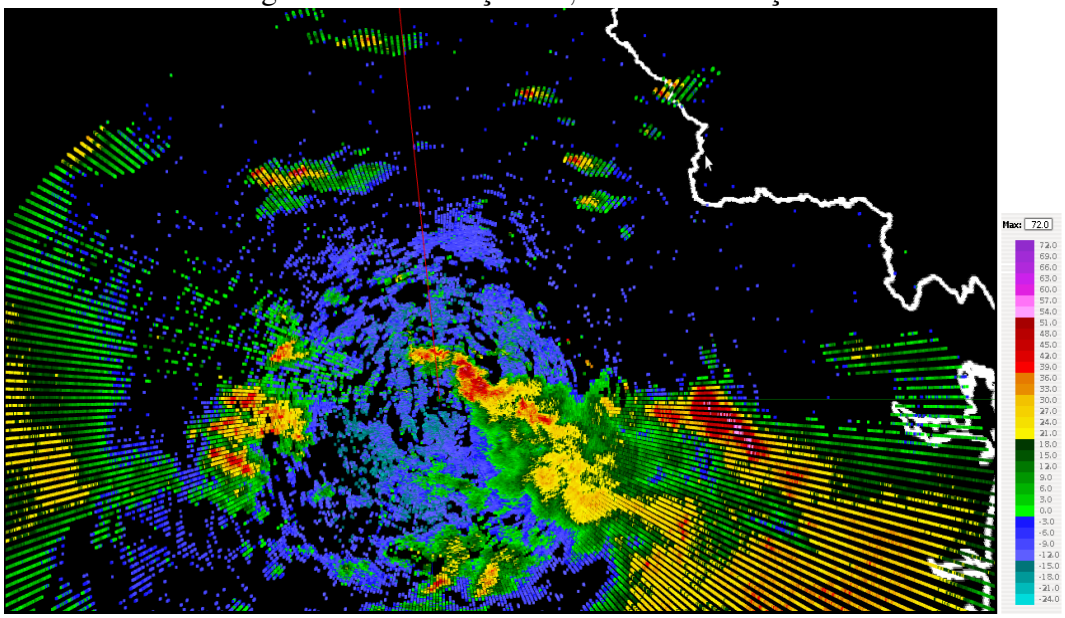
A figura 55 mostra a visualização dos dados imediatamente ao carregar o visualizador.

Figura 49: ELEVAÇÃO $0,5°$ - NAVEGAÇÃO



FONTE: O Autor

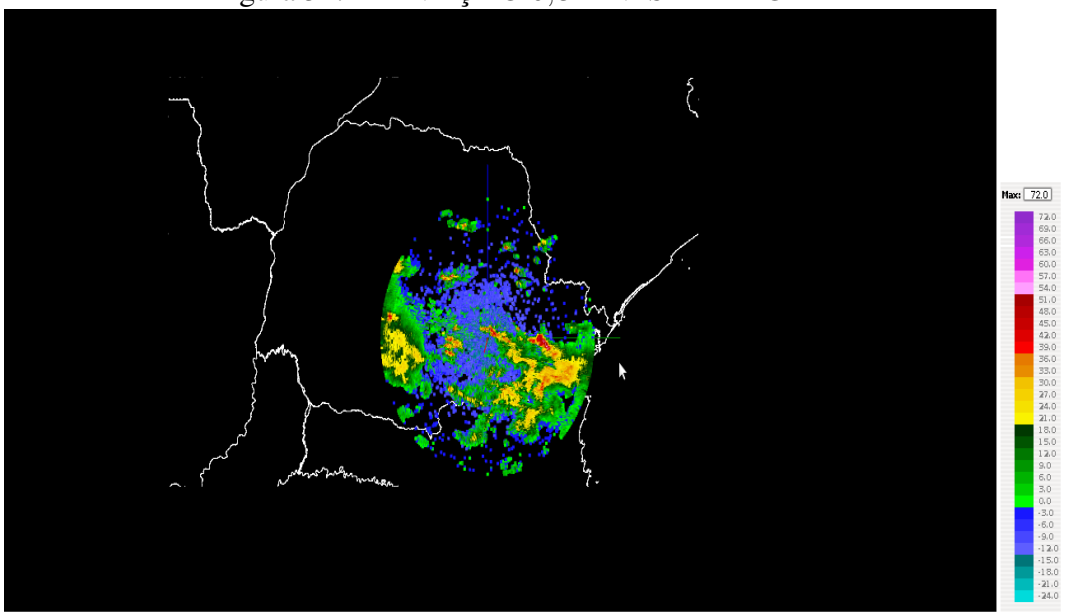
Figura 50: ELEVAÇÃO $0,5°$ - NAVEGAÇÃO



FONTE: O Autor

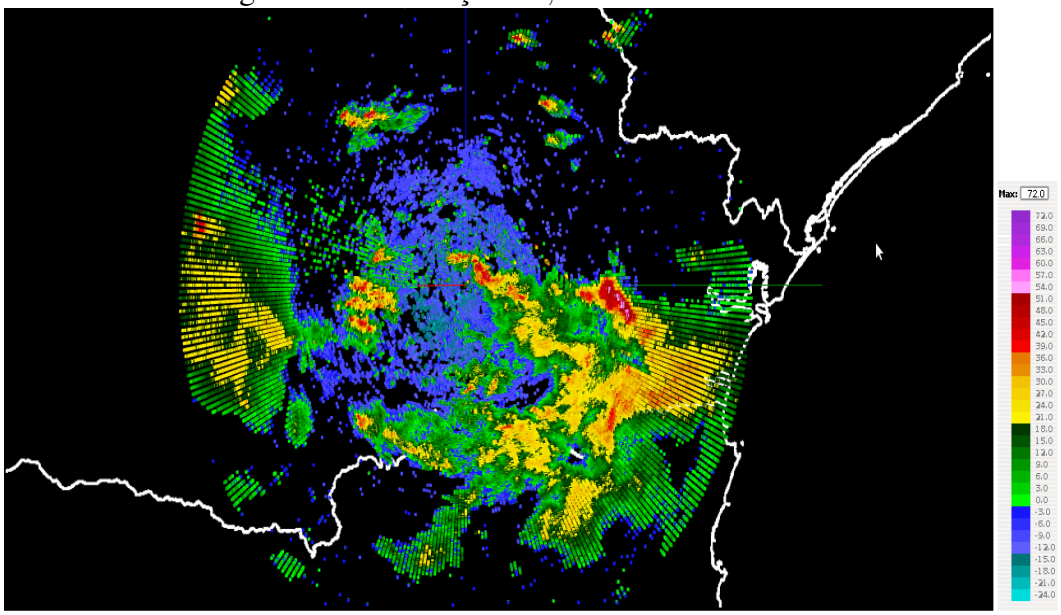
Da mesma forma que na execução anterior, foi ativada a navegação marcando a caixa de checagem “Permite navegação” (ver figura 37). As figuras 55, 56 mostram a visualização através de navegação com a teclas.

Figura 51: ELEVAÇÃO 0,5° - VISTA DE CIMA



FONTE: O Autor

Figura 52: ELEVAÇÃO 0,5° - VISTA DE CIMA



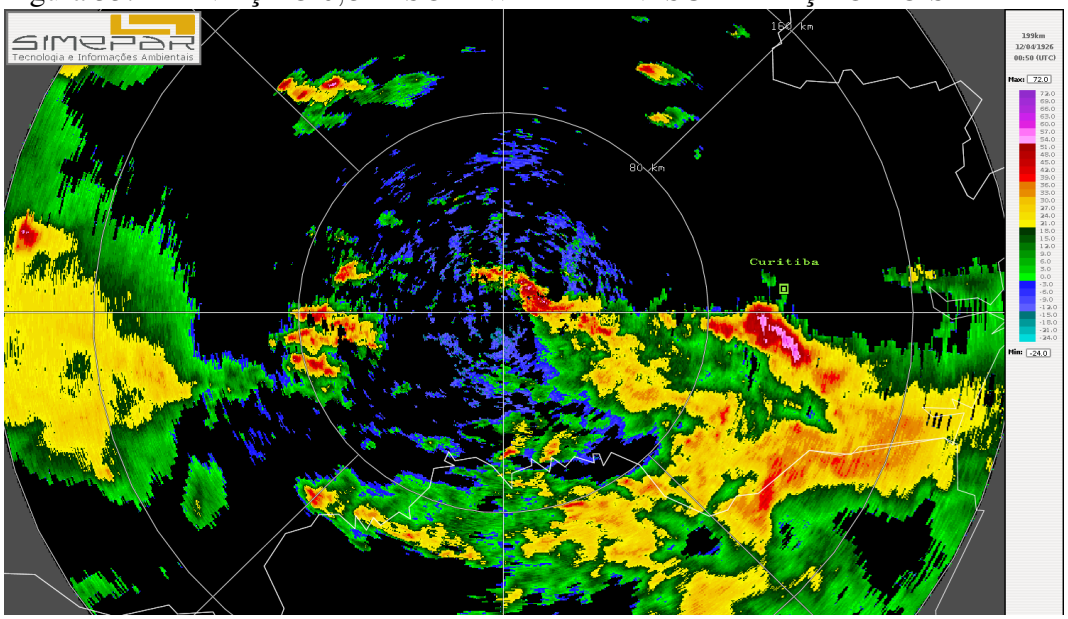
FONTE: O Autor

Elevação de 1,5°

A figura 58 mostra a visualização destes dados ao carregar o visualizador.

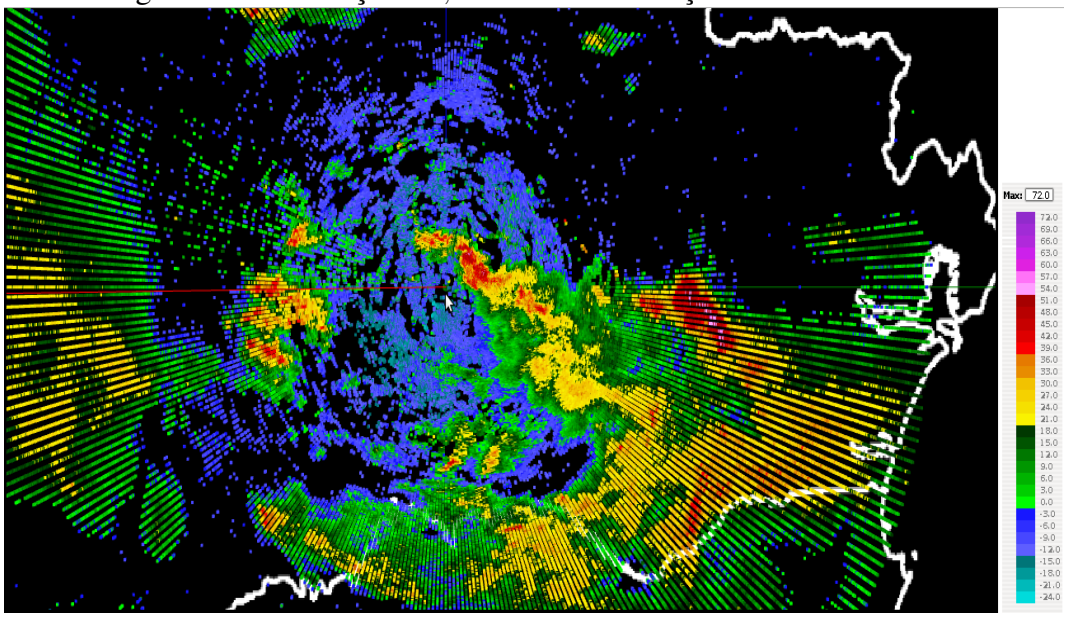
Da mesma forma que na execução anterior, foi ativada a navegação marcando a caixa de checagem “Permite Navegação” 4.2.7. A figura 59 mostra visualização com navegação nesta

Figura 53: ELEVAÇÃO 0,5° - SOFTWARE DE VISUALIZAÇÃO DO SIMEPAR



FONTE: Simepar, adaptado pelo autor

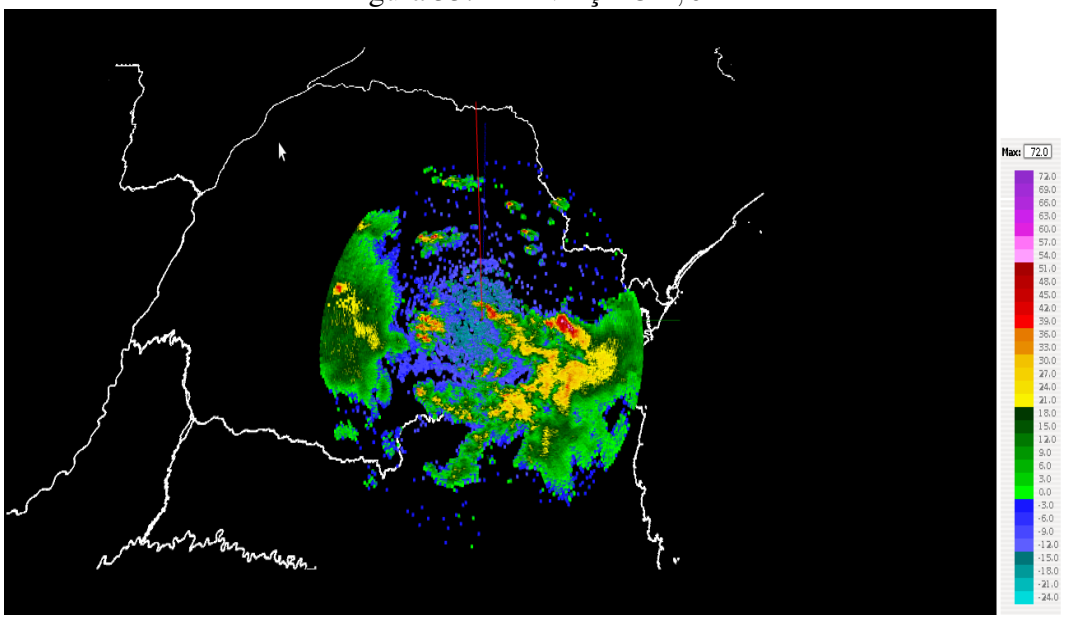
Figura 54: ELEVAÇÃO 0,5° - VISUALIZAÇÃO DE SATÉLITE



FONTE: O Autor

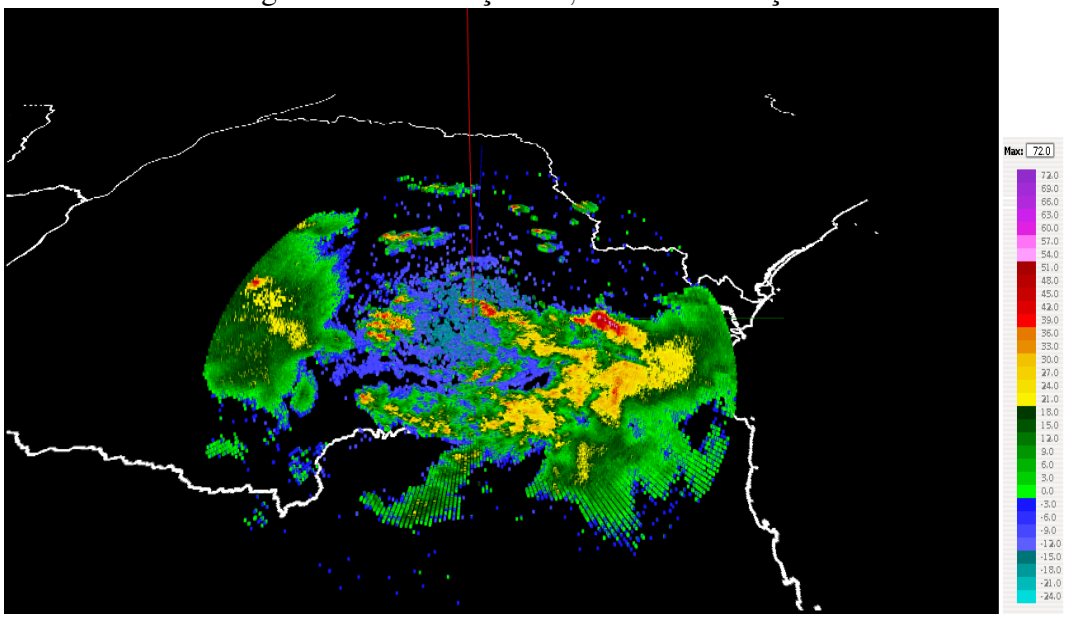
elevação.

Figura 55: ELEVAÇÃO 1,0°



FONTE: O Autor

Figura 56: ELEVAÇÃO 1,0° - NAVEGAÇÃO

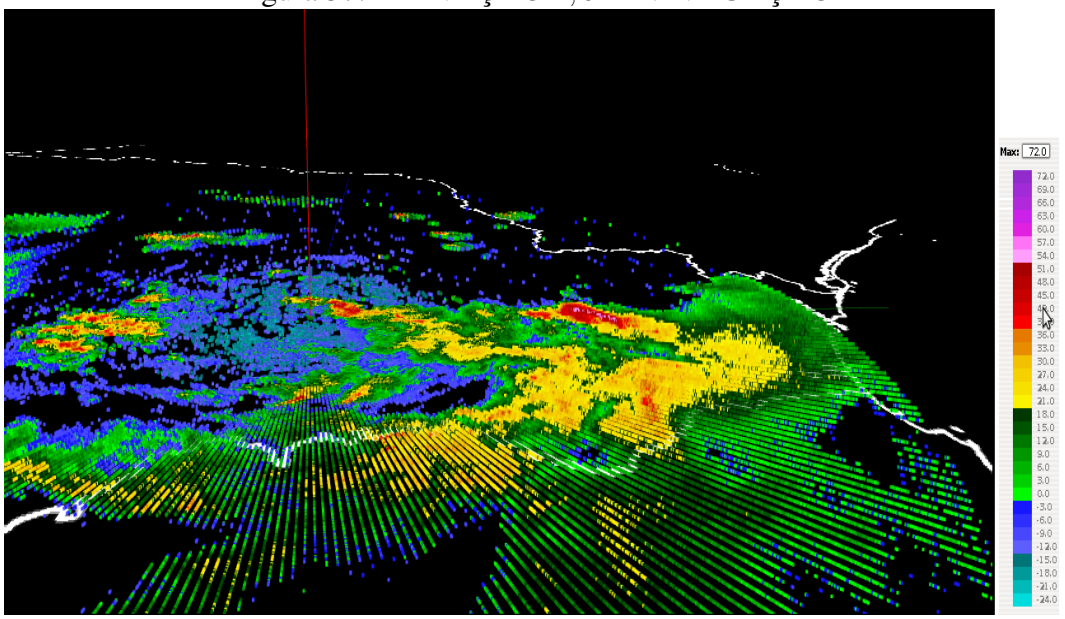


FONTE: O Autor

Elevação de 2,0°

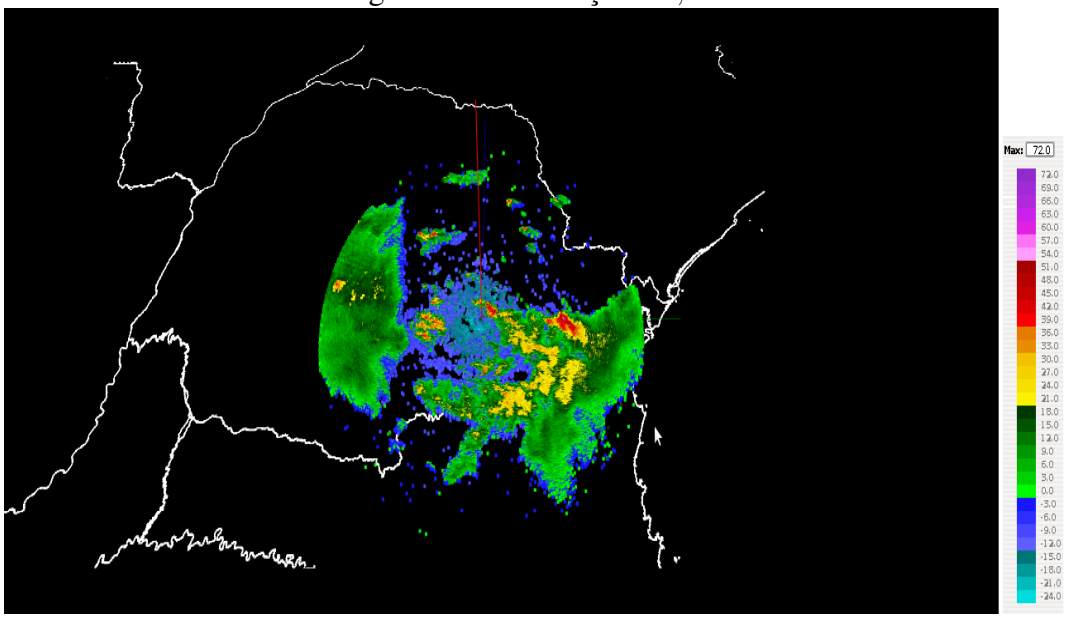
A figura 60 mostra a visualização destes dados ao carregar o visualizador. Ativando a navegação marcando a caixa de checagem “Permite Navegação” 4.2.7, tem-se a demonstração da navegação na figura 61.

Figura 57: ELEVAÇÃO 1,0° - NAVEGAÇÃO



FONTE: O Autor

Figura 58: ELEVAÇÃO 1,5°

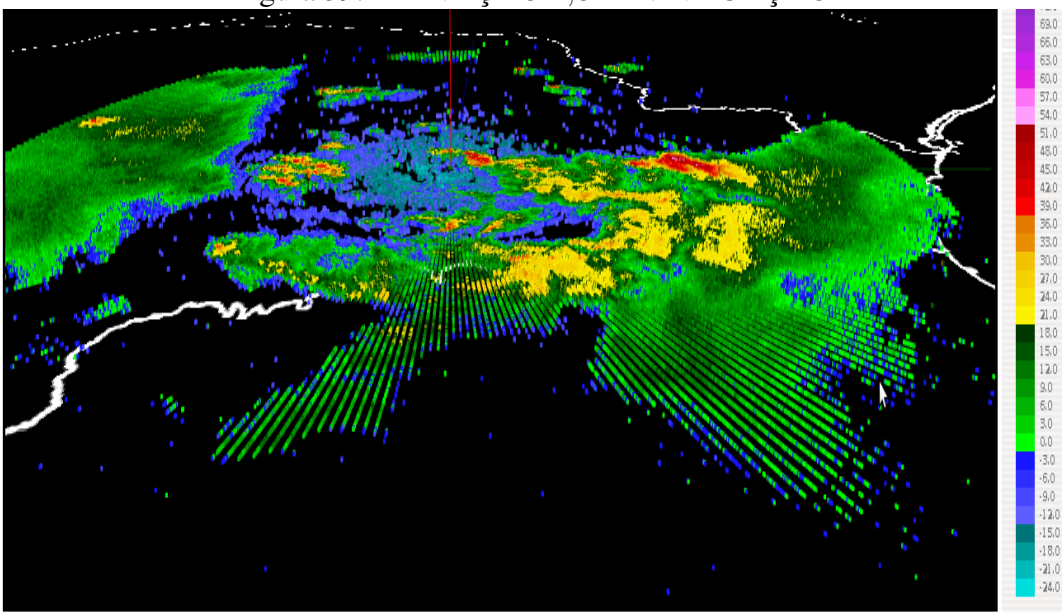


FONTE: O Autor

Elevação de 3,0°

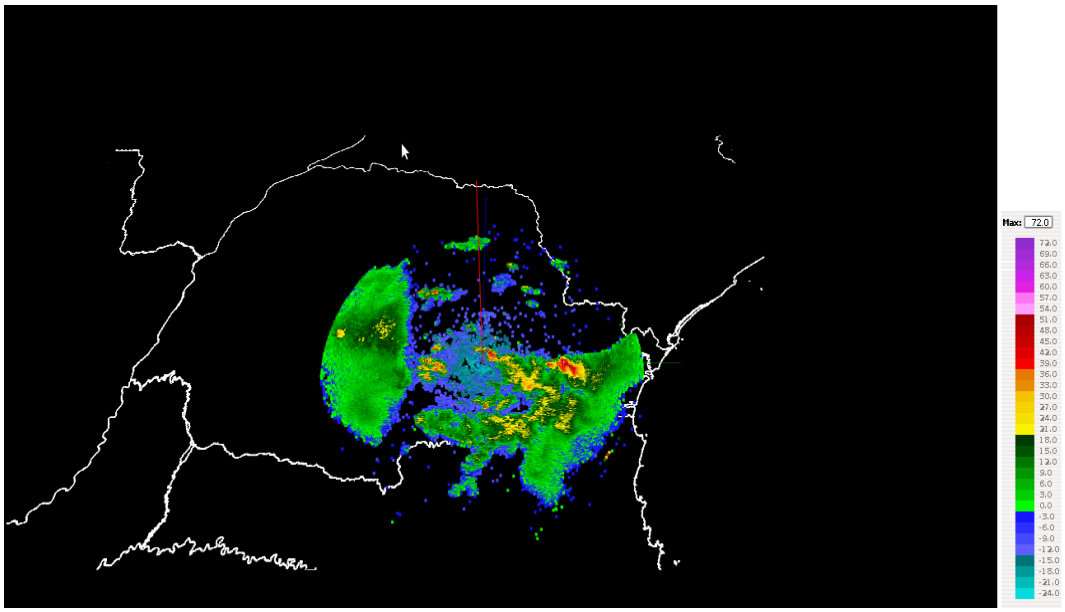
A figura 62 mostra a visualização destes dados ao carregar o visualizador.

Figura 59: ELEVAÇÃO $1,5°$ - NAVEGAÇÃO



FONTE: O Autor

Figura 60: ELEVAÇÃO $2,0°$

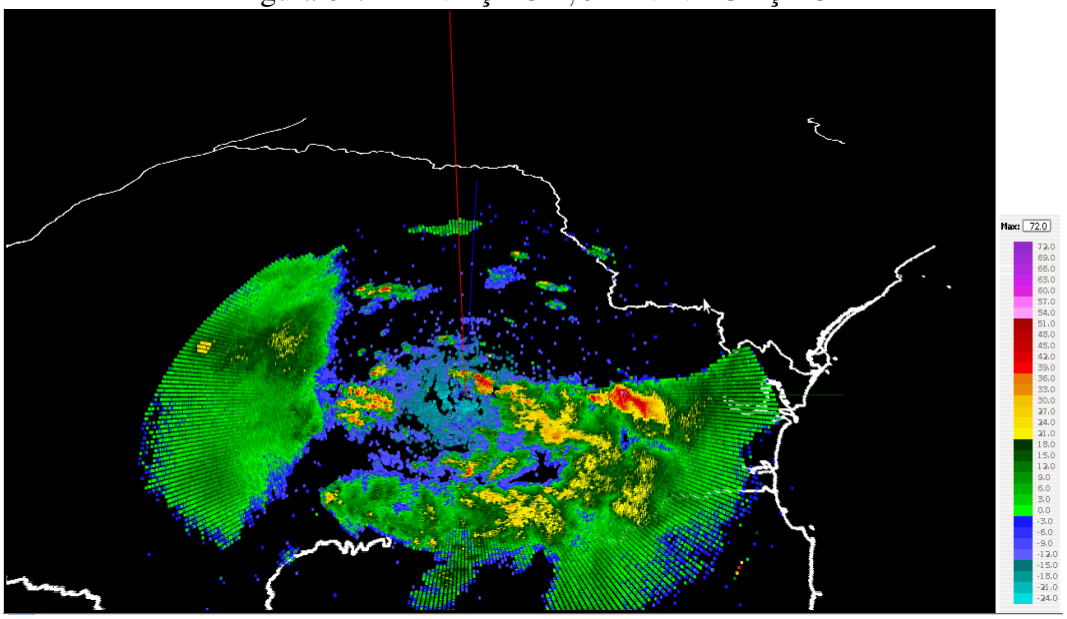


FONTE: O Autor

Elevação de $4,0°$

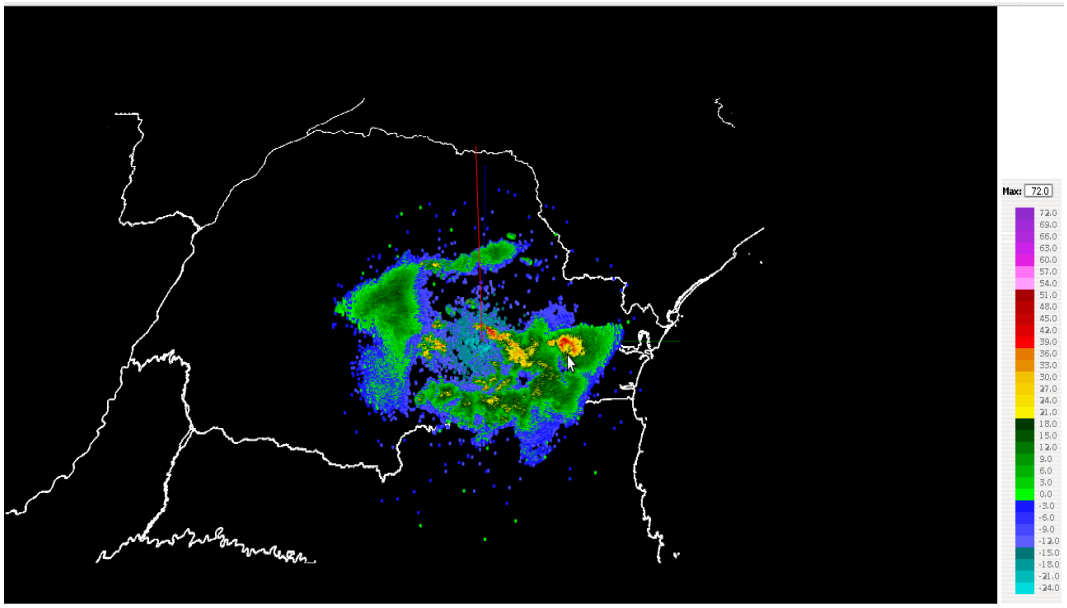
A figura 63 mostra a visualização destes dados ao carregar o visualizador. Ativando a navegação marcando a caixa de checagem “Permite Navegação” 4.2.7, pode-se navegar pelo ambiente. Posicionando a câmera bem baixo e tendo uma visão lateral, conforme demonstra a

Figura 61: ELEVAÇÃO $2,0°$ - NAVEGAÇÃO



FONTE: O Autor

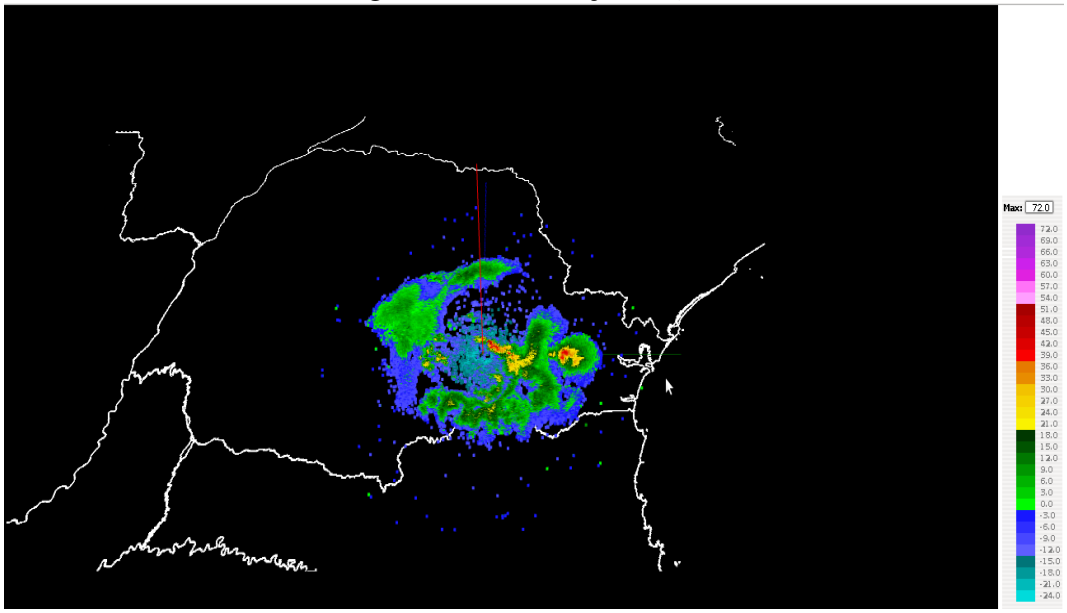
Figura 62: ELEVAÇÃO $3,0°$



FONTE: O Autor

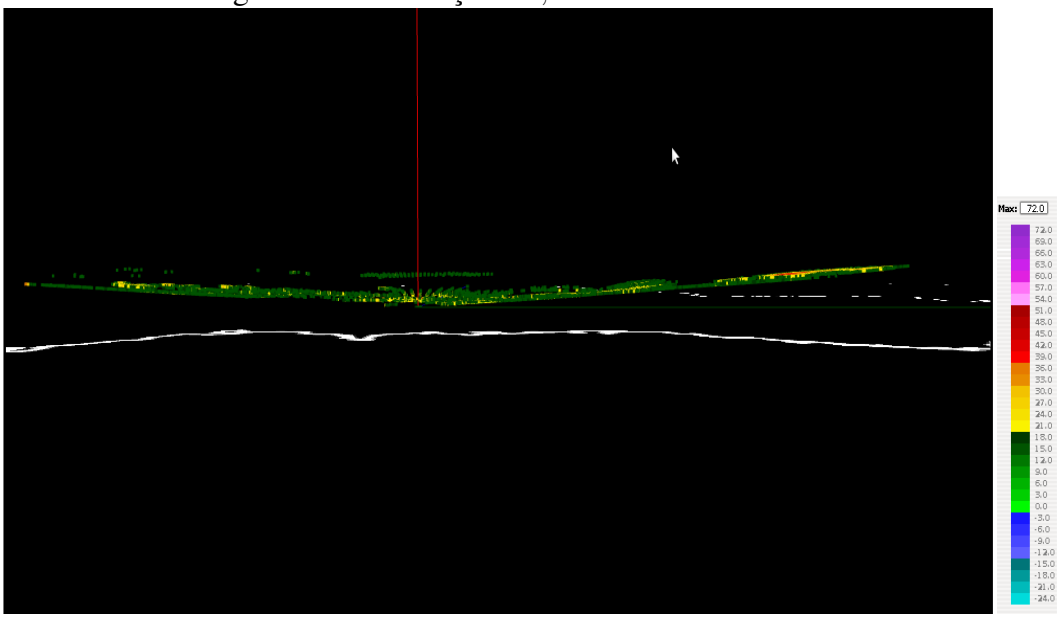
figura 64, pode-se perceber a forma de cone devido a elevação com ângulo de $4,0°$. Conforme o funcionamento do radar explicado em 2.1, as varreduras são em forma cônica, sendo percebida pelo posicionamento adequado da câmera.

Figura 63: ELEVAÇÃO $4,0°$



FONTE: O Autor

Figura 64: ELEVAÇÃO $4,0°$ - VISÃO LATERAL

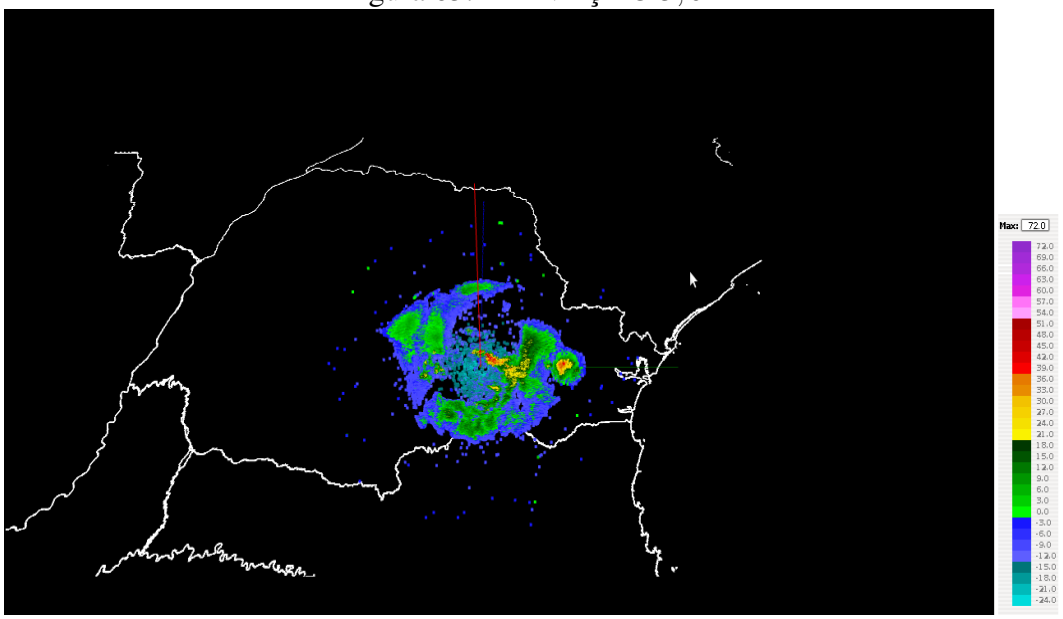


FONTE: O Autor

Elevação de $5,0°$

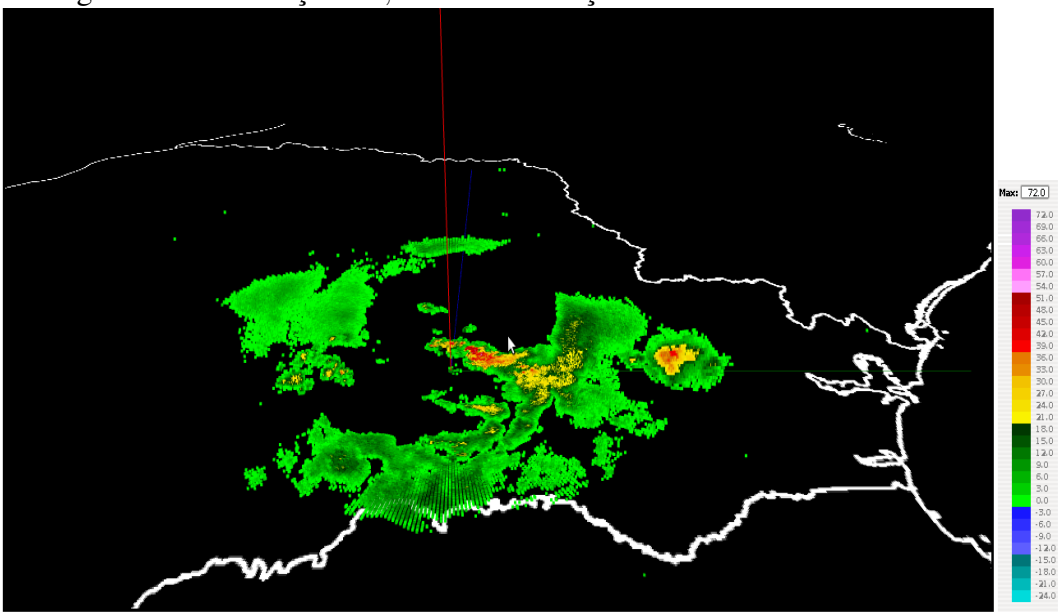
A figura 65 mostra a visualização destes dados ao carregar o visualizador. Ativando a navegação, pode-se navegar pelo ambiente. Habilitando a navegação e também aplicando-se filtro para valores entre -24 e 0. Isto é demonstrado pela figura 66.

Figura 65: ELEVAÇÃO 5,0°



FONTE: O Autor

Figura 66: ELEVAÇÃO 5,0° - NAVEGAÇÃO COM FILTRO APLICADO

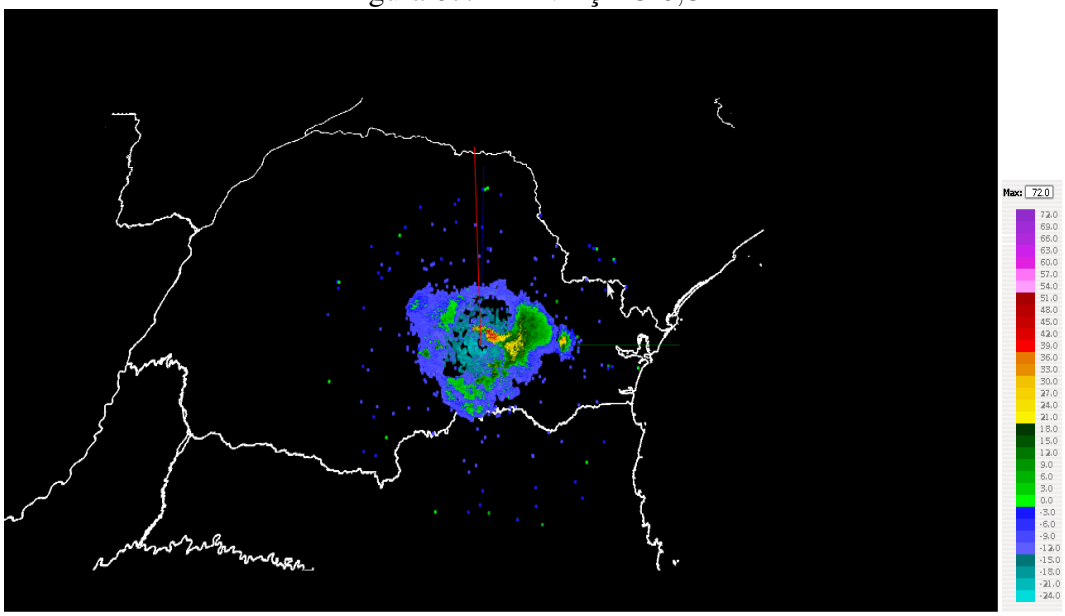


FONTE: O Autor

Elevação de 6,5°

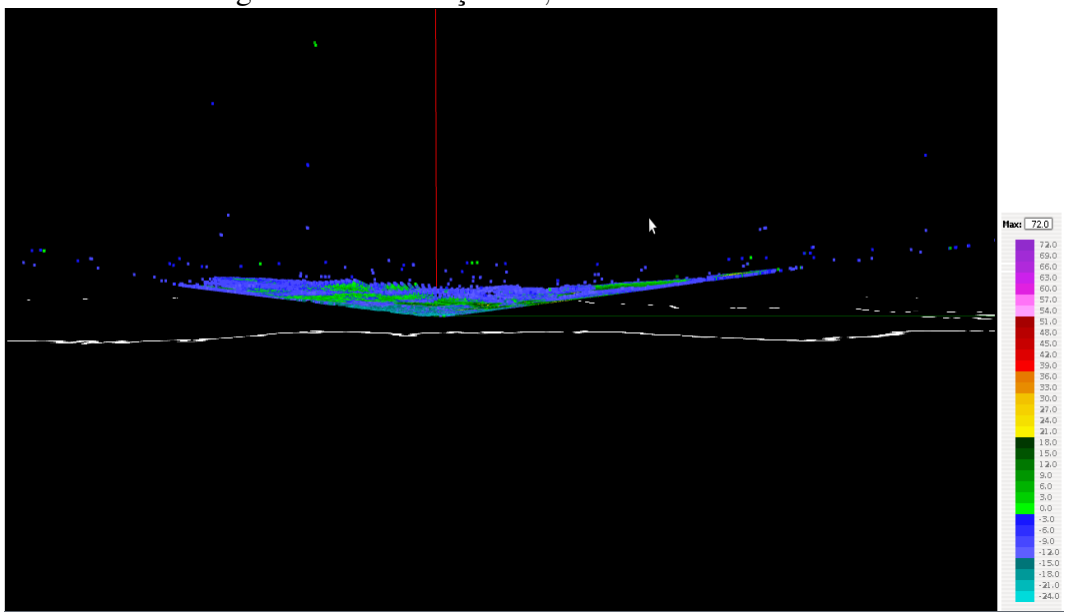
A figura 67 mostra a visualização destes dados ao carregar o visualizador. Ativando a navegação, pode-se navegar pelo ambiente e a figura 68 mostra a formação do cone.

Figura 67: ELEVAÇÃO $6,5°$



FONTE: O Autor

Figura 68: ELEVAÇÃO $6,5°$ - VISÃO LATERAL

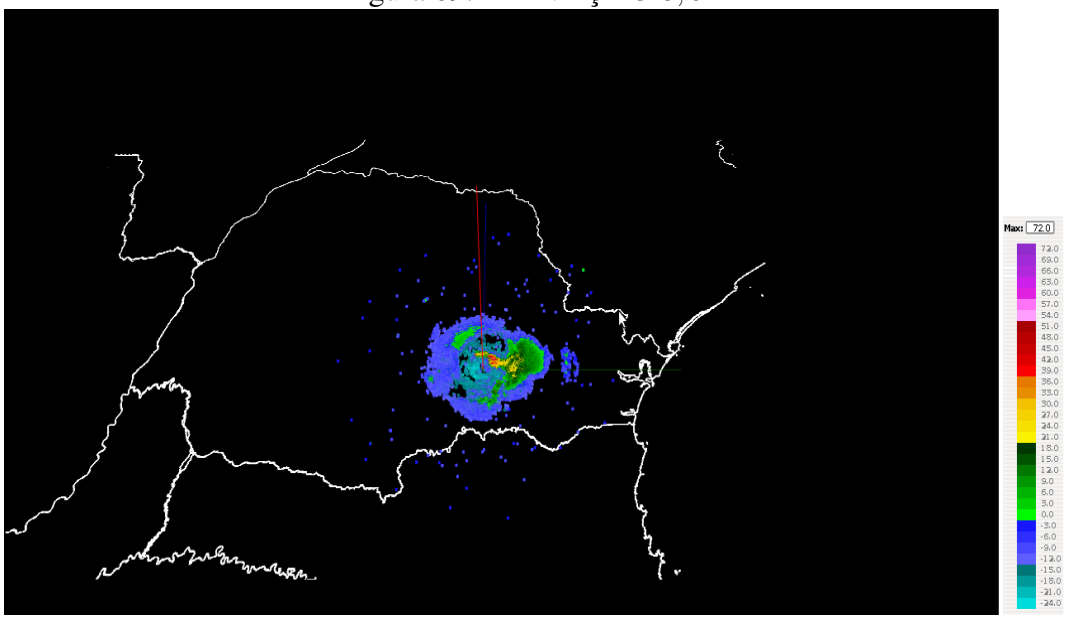


FONTE: O Autor

Elevação de $8,0°$

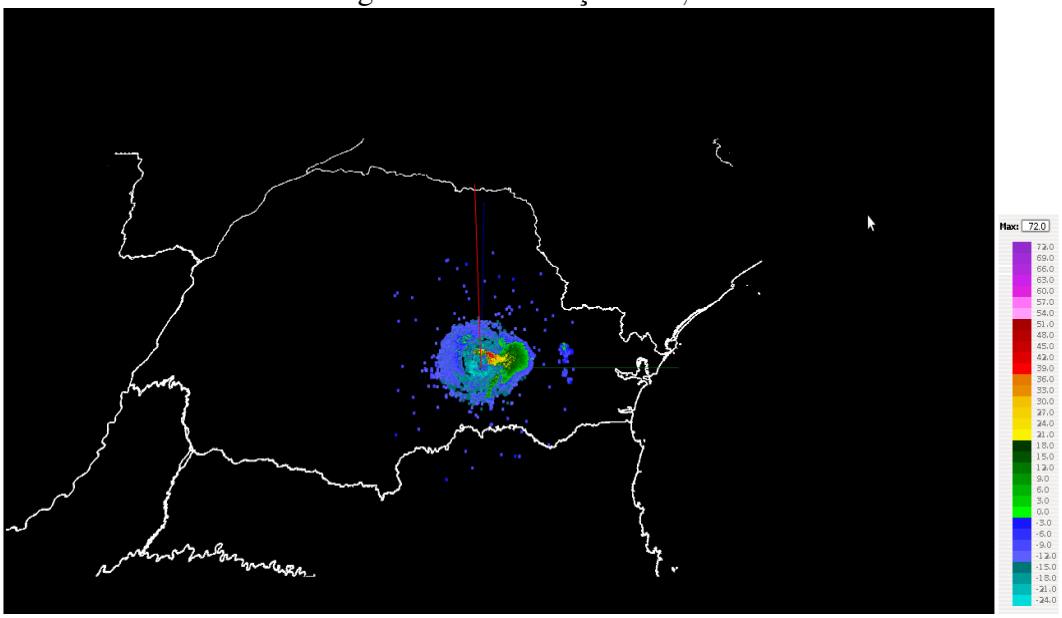
A figura 69 mostra a visualização destes dados ao carregar o visualizador.

Figura 69: ELEVAÇÃO $8,0°$



FONTE: O Autor

Figura 70: ELEVAÇÃO $10,0°$

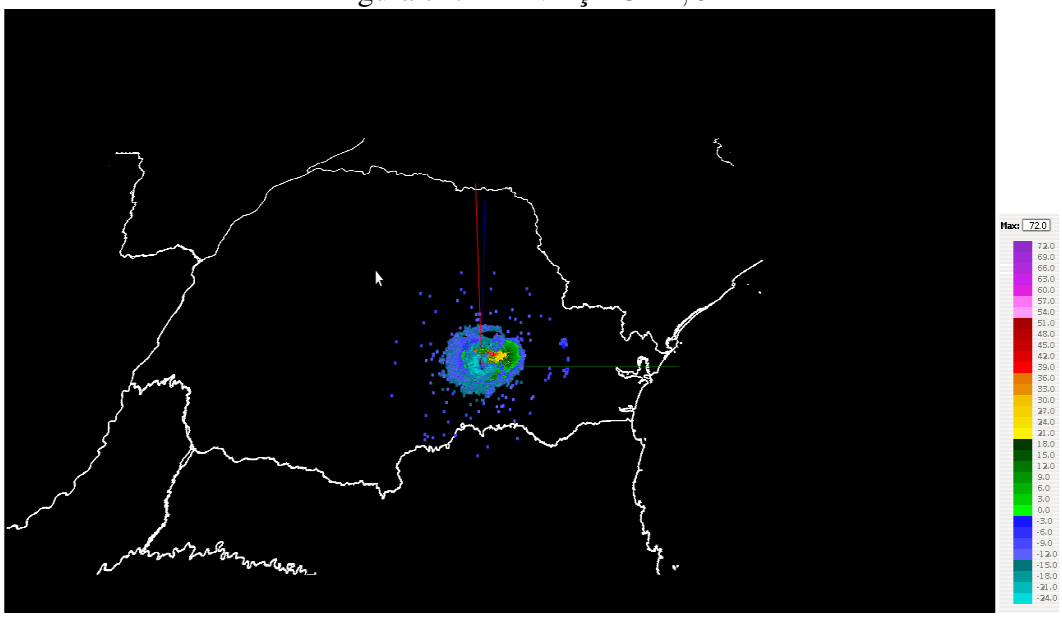


FONTE: O Autor

Elevação de $10,0°$

A figura 70 mostra a visualização destes dados ao carregar o visualizador.

Figura 71: ELEVAÇÃO $12,0°$



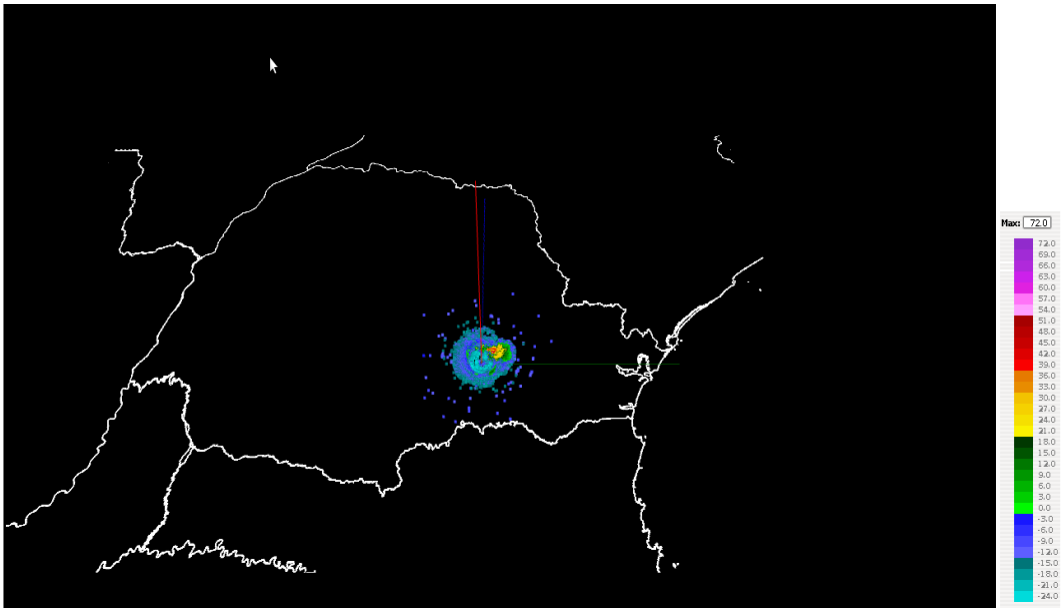
FONTE: O Autor

Elevação de $12,0°$

A figura 71 mostra a visualização destes dados ao carregar o visualizador.

Elevação de $15,0°$

Figura 72: ELEVAÇÃO $15,0°$

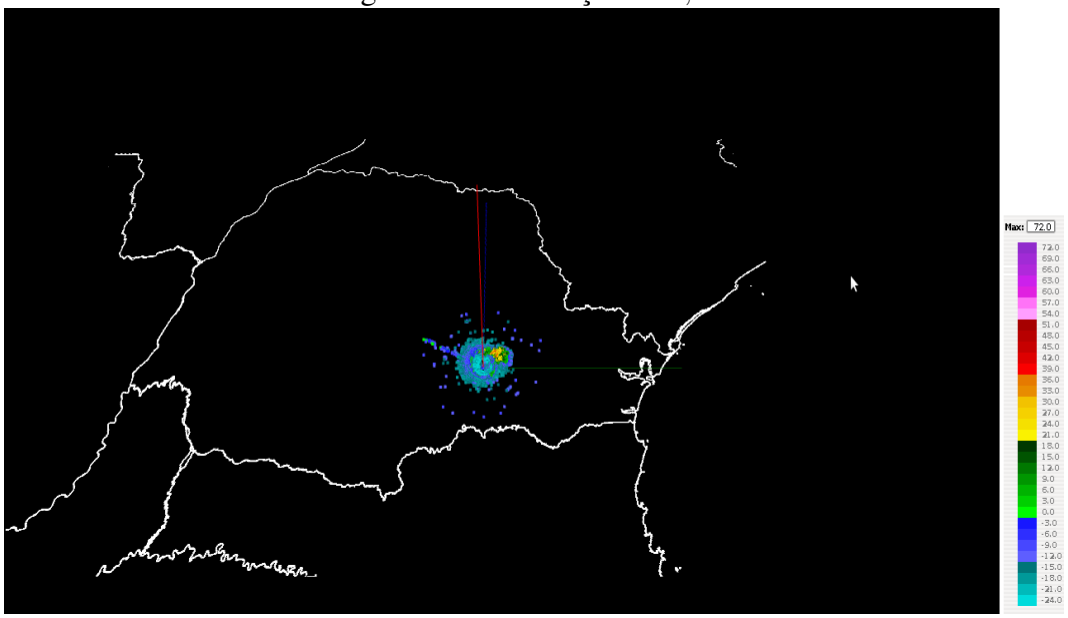


FONTE: O Autor

A figura 72 mostra a visualização destes dados ao carregar o visualizador.

Elevação de $18,0°$

Figura 73: ELEVAÇÃO $18,0°$



FONTE: O Autor

A figura 73 mostra a visualização destes dados ao carregar o visualizador.

Elevação de $21,0°$

A figura 74 mostra a visualização destes dados ao carregar o visualizador. Percebe-se claramente que a elevação cria um cone muito estreito

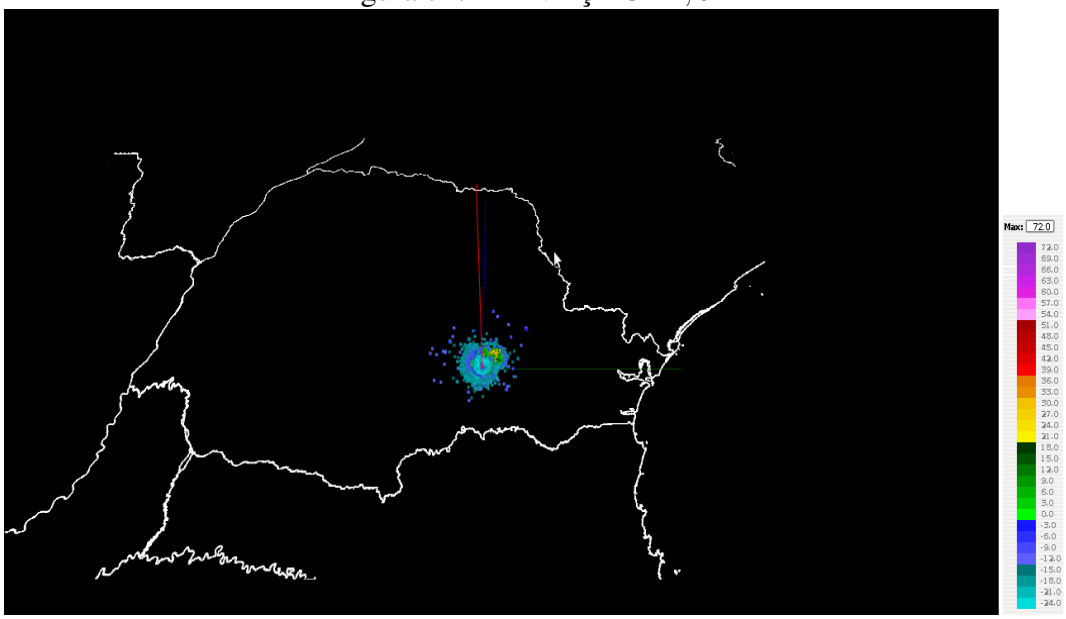
Todas as elevações

A visualização de todas as elevações constituem-se no volume de dados do radar. Este volume foi extraído e inserido no arquivo sweep_dados_14_sweeps_completo.txt.

A figura 75 mostra a visualização do dados. O conjunto de dados é muito maior que os testados anteriormente, contendo 24,8 M bytes (26.024.929 bytes). Porém, quando se habilita a navegação, percebeu-se que há a ocorrência de atrasos quando se deseja navegar.

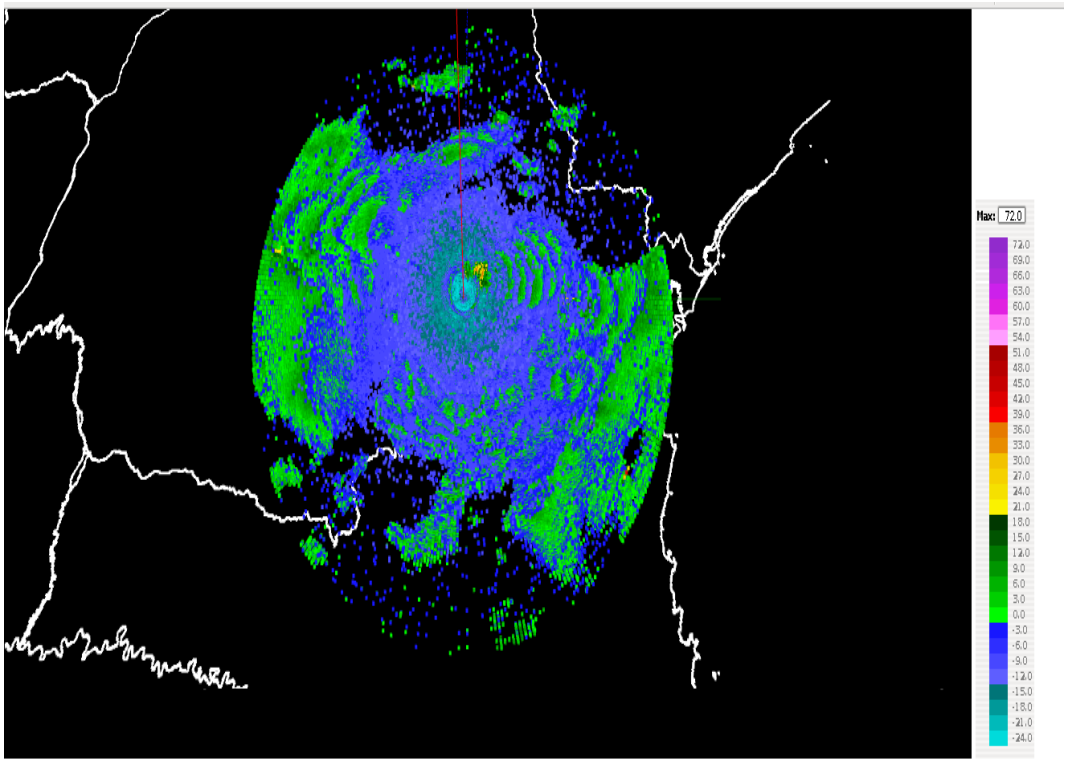
Como todas as elevações estão sendo apresentadas, há sobreposição e dependendo da posição da câmera não é possível visualizar alguns valores. Dado isto, o importante é o uso da

Figura 74: ELEVAÇÃO $21,0°$



FONTE: O Autor

Figura 75: TODAS ELEVAÇÕES

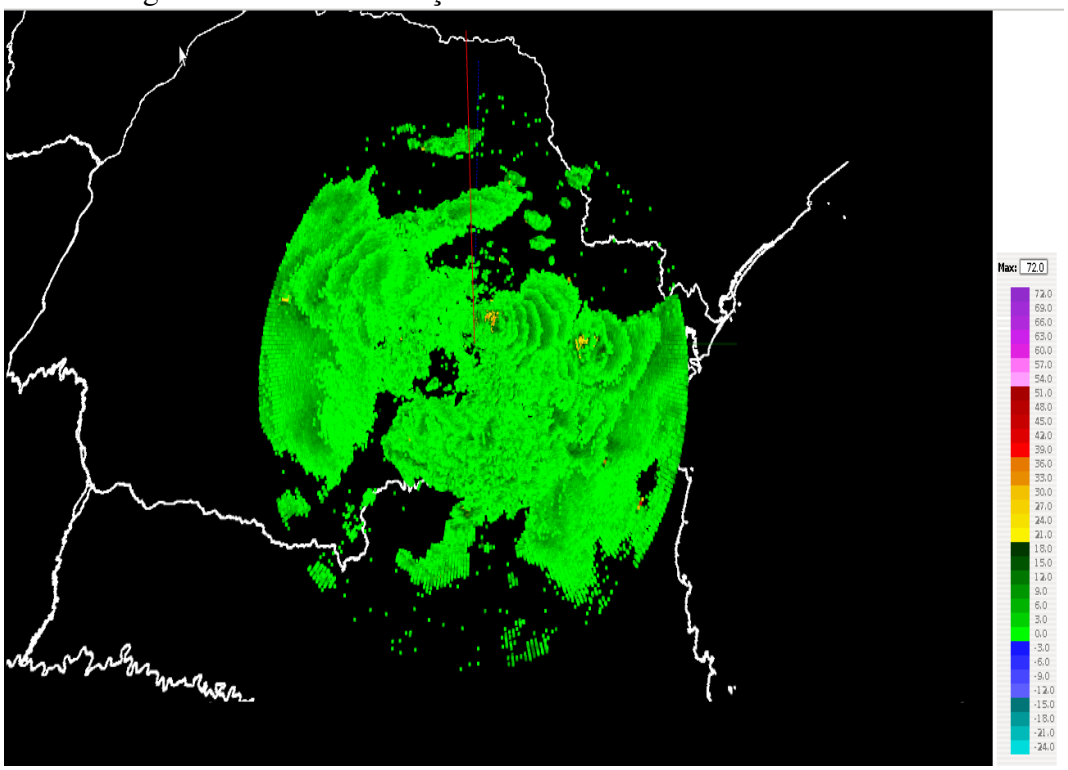


FONTE: O Autor

ferramenta de filtros para remover valores que não se desejam visualizar. A figura 76 mostra a aplicação de filtro, removendo os valores da faixa de -24 a 0 dbZ, ou seja, são mostrados valores

de refletividade acima de 0 dbZ.

Figura 76: VISUALIZAÇÃO COM DADOS ACIMA DE 0 DBZ



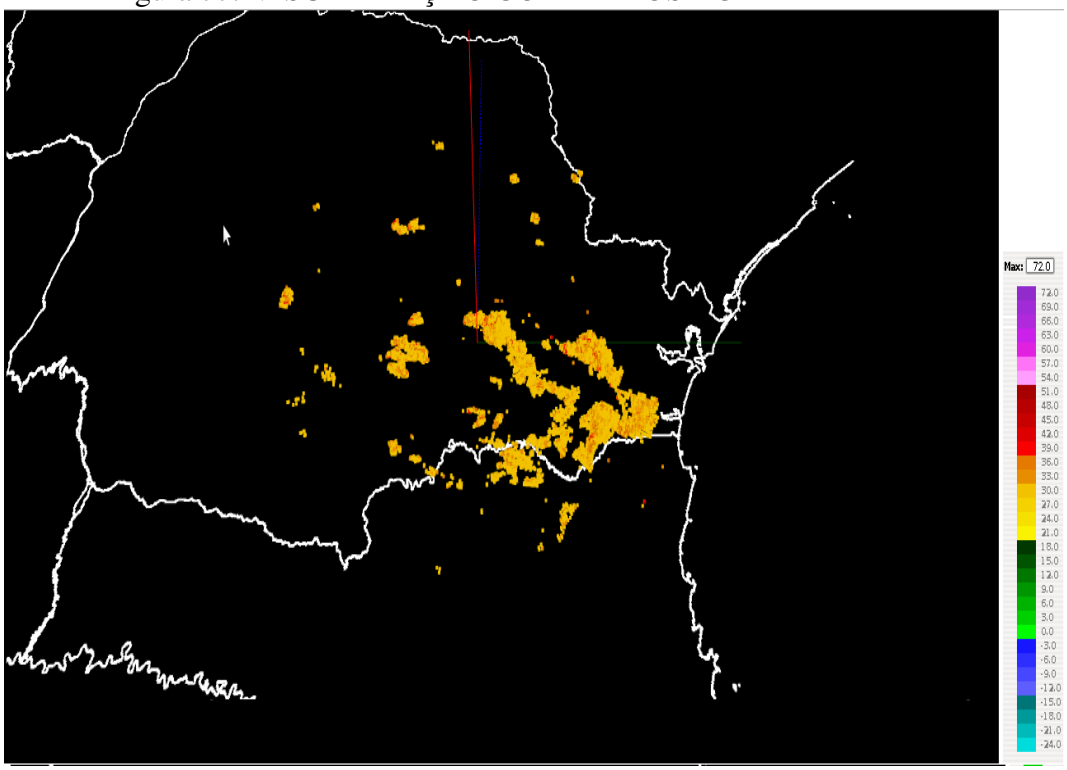
FONTE: O Autor

Mesmo com a aplicação do filtro, ainda tem-se muitos valores que encobrem outros valores. Isto se deve porque valores de chuvas de baixa intensidade aparecem na parte superior. Aplicando-se o filtro novamente, agora removendo valores -24 a 21 dbZ (ou seja, mostrando valores acima de 21 dbZ) e -24 a 30 dbZ (ou seja, mostrando valores acima de 30 dbZ), elimina-se boa parte dos valores baixos de refletividade. As figuras 77 e 78 mostram o efeito do filtro sobre os dados e sua apresentação no visualizador.

Com a aplicação do filtro, é possível perceber as células de chuva mais intensa e que são mais concentradas, conforme explica a teoria para este tipo de chuva conforme (DUTTON, 1995). Além disso, percebe-se que, com a diminuição de dados a serem apresentados, a velocidade para navegação melhorou muito, proporcionando quase sem nenhum atraso. Percebe-se pelo uso da ferramenta de monitoração do sistema [4], percebe-se claramente em alguns momentos uso intenso da CPU, alcançando valores de até 100%. Entende-se que isso se deve ao fato de que a linguagem de programação e de execução é o Javascript, sendo executado pelo interpretador embutido no próprio navegador.

[4]No Ubuntu Linux, esta ferramenta chama-se Monitor do Sistema. No Sistema Operacional Windows, a ferramenta chama-se Gerenciador de Tarefas.

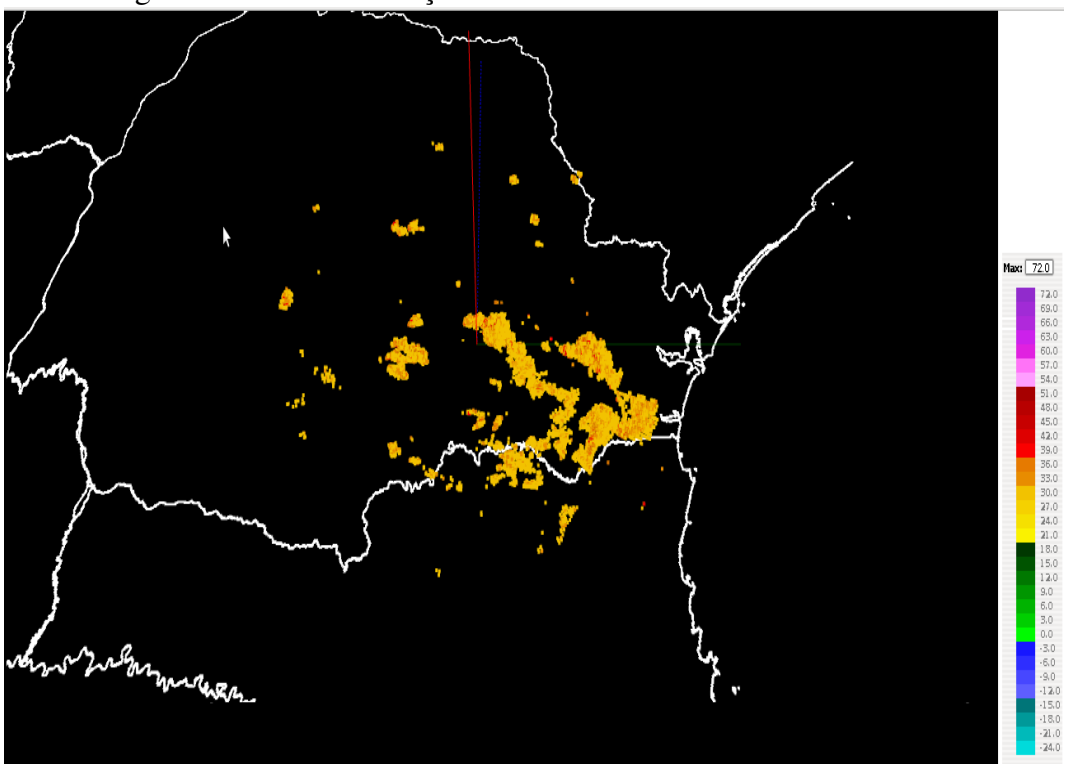
Figura 77: VISUALIZAÇÃO COM DADOS ACIMA DE 21 DBZ



FONTE: O Autor

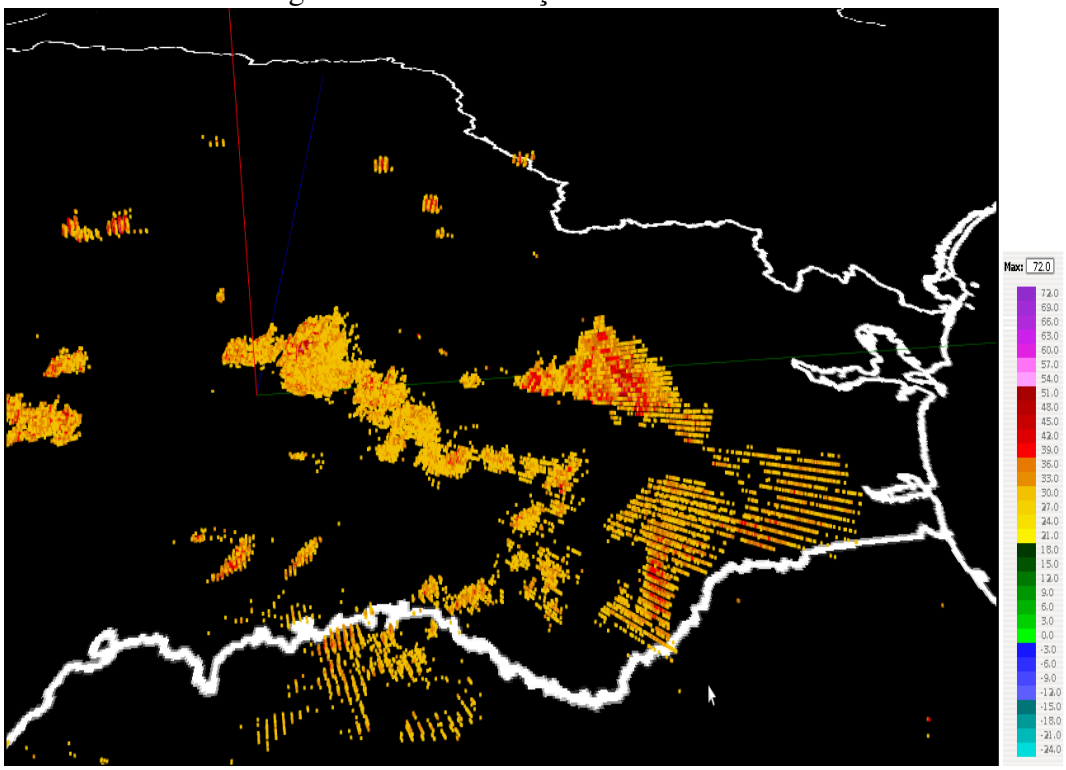
As figuras 79, 80 e 81 mostram a navegação nos dados, pelo movimento de câmera, permitindo perceber com mais clareza o fenômeno.

Figura 78: VISUALIZAÇÃO COM DADOS ACIMA DE 30 DBZ



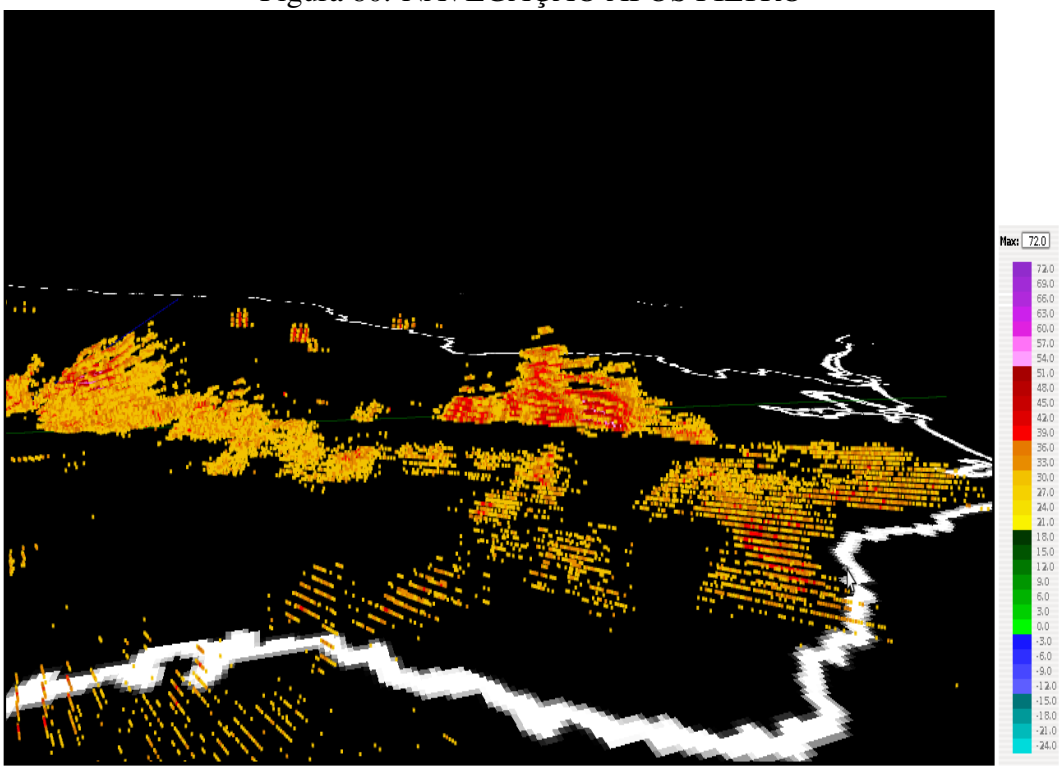
FONTE: O Autor

Figura 79: NAVEGAÇÃO APÓS FILTRO



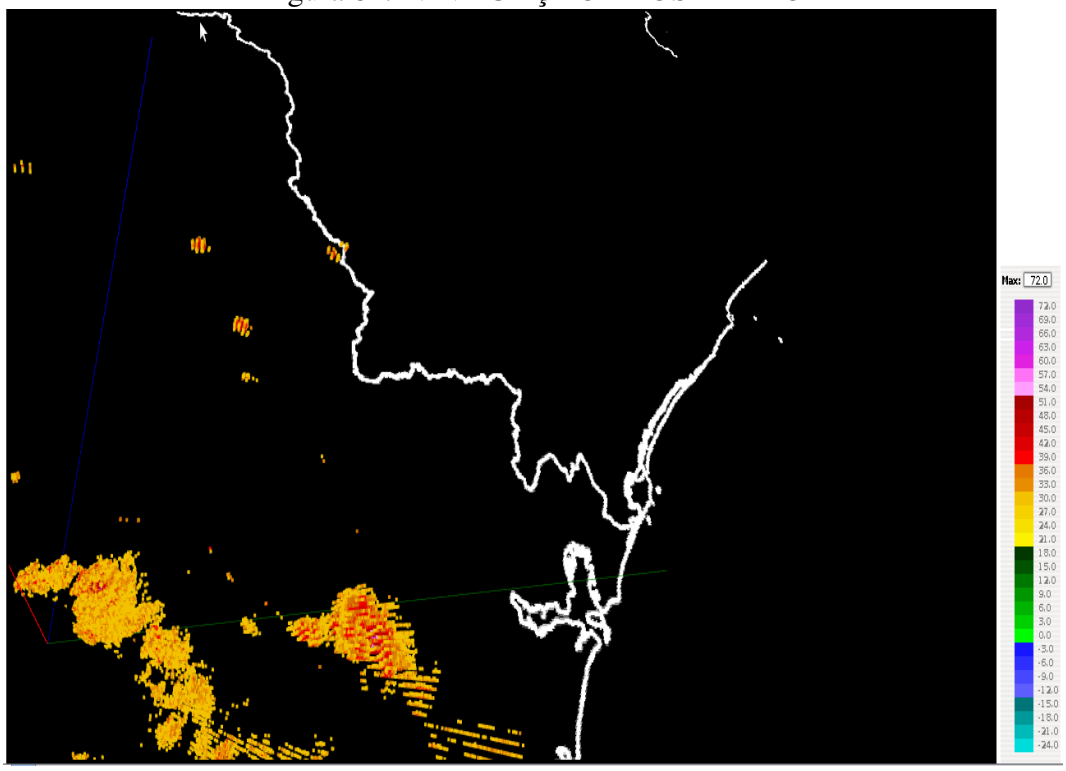
FONTE: O Autor

Figura 80: NAVEGAÇÃO APÓS FILTRO



FONTE: O Autor

Figura 81: NAVEGAÇÃO APÓS FILTRO



FONTE: O Autor

6 CONCLUSÕES

Pretendeu-se durante este trabalho, elaborar um protótipo de ferramental que permitisse a visualização tridimensional dos dados de radar meteorológico, de modo que o profissional pudesse ter acesso a visualização do fenômeno meteorológico como um todo, provendo acesso via navegador de internet a esse ferramental, porém em tempo hábil para apresentação desses dados. A navegação dentro do ambiente de visualização permitiu a observação de detalhes, como o percebido pelo sr. Reinaldo Silveira em seu depoimento. Esta melhoria na percepção do fenômeno e na forma de observação e compreensão quando comparadas com a visualização bidimensional faz parte dos objetivos específicos desse trabalho.

É possível observar que conforme as elevações vão sendo de ângulos maiores, percebe-se a formação do "cone" devido a varredura do radar, conforme explicado em 2.1. Além disto, do ponto de vista de análise do fenômeno meteorológico, é possível perceber que a intensidade da chuva, informada pela da refletividade Z varia conforme a altitude. Isto é importante para se conhecer a chuva na base e no topo da nuvem ou célula convectiva. Isto é observado na 47 comparando com a figura 74, onde as chuvas de maior refletividade são vistas em menor quantidade, predominando os valores para chuvas de intensidade baixa, entre -24 a 18 dbZ.

Na observação de todas as elevações simultaneamente, é possível observar o fenômeno físico, permitindo clareza na identificação do fenômeno. Atualmente a ferramenta bidimensional cumpre seu papel no auxílio ao profissional meteorologista nas estimativas e determinações necessárias para as previsões de nowcast. Além disto, o meteorologista é treinado durante sua formação a utilizar ferramentas bidimensionais. Porém, a visualização tridimensional permite observar o fenômeno como um todo e não em "fatias" como é feito com a ferramente bidimensional. O cerébro do profissional é que deve "montar" as fatias para que esse perceba o fenômeno total. Isto posto, entende-se que o visualizador aqui apresentado tem tremendo potencial para auxílio no uso na previsão e compreensão dos fenômenos, principalmente os fenômenos severos (chuvas intensas e tempestades).

Revisando os objetivos do trabalho conforme listados na seção 1.1 da Introdução, tem-

se:

> O objetivo geral do trabalho é desenvolver uma ferramenta de visualização de dados 3D. Ainda como objetivos específicos, tem-se:
>
> Desenvolver protótipo de ferramenta de visualização de Dados de radar meteorológico em 3 Dimensões (3D), porém com capacidade para apresentação dos dados em tempo hábil de modo que possa ser utilizado em ambiente operacional de análise de eventos meteorológicos;
>
> Esta ferramenta deverá ser capaz de visualizar os dados por meio da Internet e dentro do navegador (browser), sem nenhuma instalação de *softwares* adicionais;

A ferramenta em visualização tridimensional foi elaborada. O objetivo específico que trata que a ferramenta deve apresentar os dados em tempo hábil refere-se ao fato de que algumas ferramentas implementadas não apresentavam desempenho suficiente para uso em ambiente operacional, demorando no processo de renderização.

Baseando-se no primeiro objetivo específico, todo o desenvolvimento foi pautado em apresentar desempenho. Devido ao fato de que a codificação da aplicação ser utilizando Javascript, preocupou-se em utilizar técnicas que minimizassem o impacto no desempenho. Para tanto, escolheu-se a apresentação da "nuvem de pontos" no desenvolvimento da grade, bem como utilizando célula contendo apenas o valor do *bin* do dado do radar. Apesar disto, um volume de dados de radar tem 4.032.000 valores (*bins*), que precisam ser renderizados. Com a aplicação dos filtros dos dados, permite-se que a renderização seja mais eficaz, a partir da construção de um novo dataset a partir da filtragem dos valores no dataset original. Procurou-se ainda, não utilizar técnicas de iluminação que pudessem melhorar o aspecto estético da visualização, mas que implicam em custo computacional adicional, dado os cálculos necessários.

O visualizador aqui apresentado trabalhou adequadamente quando foi exigido para apresentar o volume de dados, ou seja, todas as elevações simultaneamente. A navegação dos dados apresentou alguns atrasos, porém pode-se atribuir a baixa capacidade de processamento da placa gráfica do equipamento utilizado para testes. Percebeu-se que quando todas as elevações (volume de dados) era apresentado, e habilitava-se a navegação no ambiente, o problema da interrupção fazia-se percebido, havendo pausas de mais de três segundos. Ao aplicar o filtro de dados, como efetuado e visto nas figuras 76, 77 e 78 esse problema era minimizado e até não sendo percebido durante a navegação. Outrossim, cabe lembrar que esse excesso de uso de CPU apenas ocorre quando a navegação está habilitada, ou seja, quando a aplicação está pronta para interatividade com o usuário.

A navegabilidade implica na construção de modelo de câmera que permita ao usuário "navegar" dentro do volume. Como WebGL não implementa o modelo de câmera, a implementação desenvolvida foi efetuada utilizando Javascript, com a construção de classes, com manipulação de matrizes para tal, conforme apresentado no Pipeline do Visualizador 4.2.3. Como se trata de ponto que poderia gerar algum tipo de comprometimento no desempenho computacional, optou-se por desenvolvimento simples e direto, mesmo com o uso de matrizes. Observa-se, como citado no capítulo 4, que existem outras técnicas, mas que não possam ser aplicadas para desenvolvimento de aplicações voltadas a ambientes operacionais, bem como não se prestam a previsão denominada "nowcast" devido à velocidade de geração das imagens.

O WebGL permite que aplicações de visualização de dados sejam construídas voltadas para a Internet. Isto mostrou-se claro, onde utilizando apenas Javascript, foi possível executar a aplicação de visualização de dados do radar meteorológico sem necessidade de *plugins*. Esta tecnologia pode então ser usada para aplicações de visualização, permitindo acesso a dados remotamente sem a instalação de outros recursos, tais como *plugins*, *applets*©, máquina virtuais ou similares.

O desenvolvimento de aplicação voltada para a Internet permite a escolha da forma em que os dados e imagens serão manipuladas, pois as aplicações são, em termos gerais, separadas em duas camadas: servidor e cliente. A camada servidor é o repositório dos dados, normalmente identificado pelo site ou portal acessado. A camada cliente é o conjunto de dados e de informações para apresentação desses dados no navegador utilizado pelo usuário. A aplicação desenvolvida em WebGL permite que todo o tratamento de dados seja feito na camada cliente sem necessidade de reconexões com o servidor, necessitando apenas quando se deseja novos dados ou informações adicionais.

Quanto à Visualização Científica, por se tratar de tecnologia nova, ainda há pouco desenvolvimento de trabalhos, conforme descrito em 3.3. Percebe-se que esta tecnologia pode ainda ser empregada para novas e melhores aplicações de visualização, principalmente ao fato de que WebGL é voltado para a Internet, não carecendo de nenhum tipo de instalação. Ou seja, aplicações que hoje necessitam de instalação e configurações, nesse ambiente apenas requerem um navegador que suporte WebGL e a conexão com a Internet. Diferente de versões de programas compilados para uma plataforma específica, onde há necessidade manter versões para cada plataforma, aplicações desenvolvidas para a Internet necessitam de apenas uma única versão de aplicação.

Apesar do uso de Javascript como sendo um fator que possa impactar no desempenho, pois a linguagem é interpretada em tempo de execução, o advento de novos computadores

disponíveis aos usuários diminuem esse problema. Outro fator de impacto para a adoção da tecnologia pode ser o uso da Internet como repositório dos dados, fator este que tem seu impacto em acesso a Internet por meio de redes com velocidade de transmissão baixas ou com dificuldades de conexão. As novas tecnologias de redes permitem que o acesso aos dados sejam melhorados, visto que se percebem melhorias na infraestrutura. Além disto, a tecnologia pode ser usada mesmo para usuários dentro de uma rede privada (rede de uma empresa, universidade ou instituto de pesquisa), não necessariamente a Internet aberta. Estes tipos de redes, usualmente possuem mecanismos de proteção, tais como *firewalls*, que impedem a instalação de programas e acesso de dados remotos sem que conexões sejam autorizadas. O fato de não necessitar de instalações de programas, permite-se que usuários possam acessar os recursos de visualização.

Entende-se portanto real potencial para construção de aplicações de Visualização Científica e também de Computação Gráfica que façam uso de WebGL como plataforma gráfica. O potencial crescimento da Internet, inclusive em aparelhos móveis, permite que esta tecnologia tenha franca expansão.

6.1 SUGESTÕES PARA TRABALHOS FUTUROS

Visando à continuidade do trabalho, propõem-se melhorias e sugestões tanto do ponto de vista acadêmico quanto para melhorar a experiência visual sob o aspecto estético e incremento nos recursos de visualização dos dados.

- Implementação de carga dinâmica dos dados - como os dados já estão em formato JSON a implementação de técnicas para carga dos dados dinamicamente está muito facilitada. O uso de AJAX pode facilitar o uso em ambiente operacional. Inclusive como sugerido pelo sr. Leonardo, podendo até aplicar filtros diretamente nos dados, visto não haver interesse em valores abaixo de 0 dbZ.

- Escalas de distâncias - Inserção de escala de distâncias para facilitar localização, principalmente quanto à altitude dos elementos.

- Interpolação - A técnica proposta neste trabalho foi o de uso de células formadas por pontos. Como a quantidade de pontos para todo o volume é 4.032.000, tem-se uma "nuvem de pontos" que permite uma boa visualização dos dados. Porém, ao aproximar-se dos dados com a navegação, percebem-se intervalos que devem ser preenchidos com a interpolação de dados. Uma proposta pode ser o uso de Radial Basis Functions (RBF)

conforme (LEWIS; PIGHIN; ANJYO, 2010), que poderiam ser aplicados aos vértices e desta forma permitir a interpolação. Um estudo foi iniciado, visando a aplicação no desenvolvimento do trabalho, mas não concluído.

O método proposto para interpolação dos dados compunha-se de, definir "células de avaliação", onde a composição de oito *bins* entre dois raios e duas elevações consecutivas, formando um volume, calcular as RBF para cada um dos *bins* e efetuar a interpolação para cada um dos "*bins* virtuais" ou "fictícios" criados e efetuar a interpolação. O procedimento para calcular esses *bins* é exaustivo, bem como o volume de dados gerados seria muito grande, dada a quantidade de novos pontos (*bins*) criados. Porém, devido a exigüidade do tempo, foi necessário decidir-se pela implementação apenas do visualizador, sem a interpolação dos dados.

- Cálculo de altitudes para permitir a identificação de altitudes de base de nuvem e topo de nuvem;
- Identificação dos pontos - cada valor precisa ser identificado por sua latitude e longitude. Isto permitiria o georreferenciamento dos dados.
- Criação de superfícies - A criação de superfícies unindo os dados permitiria uma suavização na visualização do volume, melhorando a visualização do ponto de vista estético.
- Visualização Volumétrica - Aplicando técnicas de Visualização Volumétrica, incluindo transparência melhorariam a experiência visual, porém com aumento do consumo de recursos computacionais.
- Animação ou recurso para carga de várias leituras dos dados do radar, permitindo observar a evolução dos fenômenos no tempo.

Glossário

σ_s Área de seção transversal ou scattering cross section é a área geométrica hipotética que descreve a probabilidade da luz ou outra radiação eletromagnética ser espalhada por uma partícula. 8, 9

AJAX Acrônico para Asynchronous JavaScript and XML - Javascript e XML assíncrono. Técnica utilizada na Internet para carga de dados e de código HTML de forma assíncrona, ou seja, sem necessidade da recarga de toda a página que está solicitando os dados. Desta forma, permite carga dinâmica de dados, facilitando a implementação de aplicações dinâmicas e modernas.. 101

applet Aplicativo escrito em Java que pode ser executado dentro de uma página HTML por meio da máquina virtual (JVM).. 20

bin Elemento discreto ao longo de um raio radial onde os dados são coletados. 11, 12, 31

byte code Código gerado a partir da compilação de código fonte escrito em linguagem Java. Este código é lido pela Máquina Virtual Java (ver JVM) e então executado pelo computador. Diferente de outros compiladores tais como compilador C, o compilador Java não cria código executável para um sistema operacional específico, mas cria byte code, que é executado por sua máquina virtual (JVM). 20

Chrome Navegador desenvolvido pela Google ©. 16

Firefox Navegador desenvolvido pela Mozilla ©. 16

GPU Graphics Process Unit ou Unidade de Processamento Gráfico. É o nome para a placa gráfica disponível no computador pessoal, porém tendo capacidade para manipulação de primitivas gráficas. Usualmente tem suporte as principais bibliotecas gráficas tais como OpenGL e DirectX. 22, 31

hardware Representa a parte física de computadores: placas gráficas, placas de rede, discos para armazenamento, etc. 18

Internet Rede Mundial de Computadores, utilizada por todo mundo para disseminação de informação, pesquisa científica, entretenimento, comunicação. 19

JDK Java Developers Kit. Conjunto de programas e utilitários que permitem a compilação e execução de programas escritos com linguagem Java. Também é utilizado por Servidores de tecnologia JSP que necessitam compilar e executar código Java. Exemplo de servidor é o Apache Tomcat.. 76

JVM Java Virtual Machine ou Máquina Virtual Java: ambiente que interpreta a código compilado Java (Java Byte Code) e o executa. Permite que um código Java seja compilado apenas uma vez, mas executado em várias plataformas, bastando apenas que a máquina virtual esteja instalada e execute o código compilado em Java ByteCode.. 20

navegador Programa (software) que permite o acesso a páginas na Internet. São exemplos de navegadores o InternetExplorer ©, Mozilla Firefox ©, Google Chrome ©, Safari ©, entre outros.. 16, 19, 20, 28, 32, 35, 37, 56, 73, 76, 93, 100

Nowcast Previsão de curto espaço de tempo normalmente de 0 a 6 horas. 2

OpenGL Conjunto de instruções (api)para programação gráfica para 2D e 3D, desenvolvida pelo KhronosGroup. 17–20

OpenGL ES Conjunto de instruções (api) para programação gráfica para 2D e 3D para sistemas embarcados, tais como consoles, telefones celulares, dispositivos de veículos.
Constituida por um sub conjunto das instruções disponibilizadas em OpenGL. Desenvolvido e mantido pelo KhronosGroup. 16, 18, 19

OpenGL ES SL Shading Language (a tradução literal é "linguagem de sombreamento") para o ambiente com menor capacidade de processamento e memória, tais como celulares e tablets. 16, 22, 25

OpenGL SL OpenGL Shading Language ou Linguagem de Shaders OpenGL - a tradução literal é "linguagem de sombreamento".
Conjunto de instruções de alto nível para programação dos Shaders de Vértices e Shaders de Fragmento. 18

pipeline Diagrama contendo sequência de unidades funcionais que executam uma tarefa em vários estágios. Cada estágio obtém dados de entrada e produz um resultado. Este resultado é então, utilizado como dado de entrada para o próximo estágio. Também pode ser expresso com Diagrama em Blocos. 18, 32

plugin Programa que permite ser embutido em outro, adicionando funcionalidades a este último. 16, 19, 20

radar Radar é um acrônimo em inglês para “Radio Detection and Ranging”; tradução literal “Detecção e estimativa de distância por rádio”, é equipamento que transmite ondas eletromagnéticas e capta parte do sinal refletido ou ecoado por obstáculos. Com este eco, pode detectar obstáculos, bem como estimar a posição e distância a partir do transmissor . 1–4, 6, 11–13

render O verbo “to render” do idioma inglês significa literalmente “manter alguém em um estado particular”. Na Computação Gráfica, significa apresentar imagens em um monitor de vídeo ou outro dispositivo de saída, de modo que os elementos gráficos sejam representados (RENDER,). Em português, o verbo é adaptado e usa-se o termo *renderizar*. 31, 32, 105

renderizar Renderizar é o uso do verbo “render” - ver render. 28, 30

renderização Uso do verbo render na língua portuguesa, para explicar o sentido de apresentação de imagem em dispositivo de saída - ver render. 19, 31, 65–67, 73, 99

Safari Navegador desenvolvido pela Apple ©. 16

shader a tradução literal é “sombreador”; Pequena porção de código para programação da placa gráfica ou GPU, substituindo alguma funcionalidade fixa provida pela biblioteca gráfica, permitindo que o desenvolvedor tenha maior controle e flexibilidade sobre a criação na renderização dos elementos primitivos. O termo foi cunhado pela empresa Pixar©e atualmente é utilizado pelas principais bibliotecas de computação gráfica: OpenGl e Direct3D (SHADER,). 16, 18, 23, 65

tronco de visão Tronco de pirâmide formado pela porção da pirâmide de visão delimitada pelos plano anterior e posterior (Bruno Eduardo Madeira, 2006, p.21). Também é denominado na literatura com Viewing Frustum. 23

Volume de Visão Volume formado pela porção da pirâmide de visão delimitada pelos plano anterior e posterior (FOLEY et al., 1990, p.241). Recebe também o nome de Pirâmide Visão ou Tronco de Visão. 23

WEB Relativo ou relacionado com a Internet. Literalmente, significa “teia de aranha”. 16

Referências Bibliográficas

ALONSO, M.; FINN, E. *Física um curso universitário*. São Paulo Brasil: Edit. Edgard Blucher, 1972.

ANTTONEN, M. et al. Transforming the web into a real application platform: new technologies, emerging trends and missing pieces. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2011. (SAC '11), p. 800–807. ISBN 978-1-4503-0113-8. Disponível em: <http://doi.acm.org.ez22.periodicos.capes.gov.br-/10.1145/1982185.1982357>.

ANYURU, A. *Professional WebGL Programming: Developing 3D Graphics for the Web*. USA: Wrox, 2012. Disponível em: <http://media.wiley.com/product_data/excerpt/60-/11199688/1119968860-94.pdf>.

BLYTHE, D. *OpenGL ES Common Profile Specification*. 2002. Disponível em: <http:-//www.khronos.org/registry/gles/specs/1.0/opengles_spec_1_0.pdf>. Acesso em: 12 apr 2012.

Bruno Eduardo Madeira. *Calibração Robusta de Vídeo Para Realidade Aumentada*. Dissertação (Mestrado) — Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, 18 dez. 2006.

BULL, J. M. et al. Benchmarking java against c and fortran for scientific applications. In: *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*. New York, NY, USA: ACM, 2001. (JGI ́01), p. 97–105. ISBN 1-58113-359-6. Disponível em: <http://doi.acm.org/10.1145/376656.376823>.

BUNDERI, R. *The Invention That Changed the World: How a Small Group of Radar Pioneers Won the Second World War and Launched a Technological Revolution*. 1230 Avenue of the Americas, New York, New York: Touchstone Books, 1997.

CANTOR, D.; JONES, B. *WebGL Beginner's Guide*. Livery Place 35 Livery Street Birmingham B3 2PB, UK: Packt Publishing, 2012.

CAPES. *Site de Periódicos*. 2012. Disponível em: <http://periodicos.capes.gov.br>. Acesso em: 10 mar. 2012.

CASTELLANO, M. S.; NUNES, L. H. Avaliação espacio-temporal das precipitações extremas e seus impactos no meio urbano: um caso brasileiro. *Territorium Revista Portuguesa de Riscos, Prevenção e Segurança*, Universidade de Coimbra, v. 17, p. 35–44, 2010. Disponível em: <http://www.uc.pt/fluc/nicif/riscos/Documentacao/Territorium/T17_artg/05Territorium_35-44-.pdf>.

CESAR JR, R. M. *Reconstrução Tridimensional por Ajuste de Superfícies Paramétricas*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 1994.

CROCKFORD, D.; IETF. *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006. RFC 4627. Disponível em: <http://www.ietf.org/rfc/rfc4627-.txt?number=4627>. Acesso em: 08 Mai. 2012.

DOGGETT IV, A. L. et al. 3D Visualization of Weather Radar Data. In: AMS. [S.l.]: AMS, 2002. 21st Conf. on Severe Local Storms.

DUTTON, J. A. *Dynamics of Atmospheric Motion*. New York USA: Dover Publications Inc, 1995.

ECMA. *ECMAScript Language Specification 262*. 2011. Disponível em: <http://www-.ecma-international.org/publications/standards/Ecma-262.htm>. Acesso em: 07 Mai. 2012.

ERNVIK, A. *3D Visualization of Weather Radar Data*. Dissertação (Mestrado) — Linköping University, 2002.

FOLEY, J. D. et al. *Computer Graphics Principles and Practice*. 2nd ed. ed. [S.l.]: Addison Wesley Publishing Company, 1990. ISBN 020112110-7.

FOLHA DE SÃO PAULO. Número de mortos na região serrana do rio passa de 900. *Folha de São Paulo*, 16 fev. 2011. Cotidiano. Disponível em: <http://www1.folha.uol.com.br/cotidiano-/876441-numero-de-mortos-na-regiao-serrana-do-rio-passa-de-900.shtml>. Acesso em: 12 abr. 2011.

FOLHAPRESS. Inundações afetam mais de 200 mil pessoas na austrália. *Gazeta do Povo*, 4 jan. 2011. Clima.

GOMES, J.; VELHO, L. *Fundamentos da Computação Gráfica*. Rio de Janero: IMPA, 2008. ISBN 978-85-244-0200-5.

GRACIA JAVIER ; BAYO, E. An Integrated 3D Web Application for Structural Analysis Software as a Service. *American Society of Civil Engineers*, p. 30, abr. 2012. ISSN 1943-5487. Disponível em: <http://dx.doi.org.ez22.periodicos.capes% -.gov.br/10.1061/(ASCE)CP.1943-5487.0000217>. Acesso em: 16 apr 2012.

HILL JR, F. S. *Computer Graphics: using OpenGL*. 2nd. ed. Upper Saddle River, N.J.: Prentice Hall, 2001. ISBN 0023548568.

HO, S. *Homogeneous Coordinates*. 2012. Http://www.songho.ca/math/homogeneous/homogeneous.html. Disponível em: <http:/-/www.songho.ca/math/homogeneous/homogeneous.html>. Acesso em: 08 Ago 2012.

HO, S. *OpenGL Transformation*. 2012. Http://www.songho.ca/opengl/gl_projectionmatrix.html. Disponível em: <http://www-.songho.ca/opengl/gl_projectionmatrix.html>. Acesso em: 08 Ago 2012.

IEEE. *Albert H. Taylor*. 2012. Disponível em: <http://www.ieeeghn.org/wiki/index.php-/Albert_H._Tayl>. Acesso em: 15 jul. 2011.

INMETRO. *Velocidade da Luz*. 2012. Disponível em: <http://www.inmetro.gov.br/infotec-/publicacoes/Si.pdf>. Acesso em: 07 fev. 2012.

INTRODUCING JSON. 2012. Disponível em: <http://www.json.org>. Acesso em: 07 Mai. 2012.

KhronosGroup Inc. *Webgl specification*. 2011. 12 p. Disponível em: <https://www.khronos-.org/registry/webgl/specs/1.0>. Acesso em: 10 mar. 2011.

KhronosGroup Inc. *History of OpenGL*. 2012. Disponível em: <http://www.opengl.org/wiki-/History>. Acesso em: 11 feb 2012.

KNOX, K. J. *Light-Induced Processes in Optically-Tweezed Aerosol Droplets*. Berlin: Springer, 2011. (Springer Theses). ISBN 9783642163470.

KRAUSE, U. M. M. abr. 2000. Disponível em: <http://www.airpower.maxwell.af.mil-/airchronicles/bookrev/buderi.html>.

LANDEMAN, R. W. *Computer Graphics: 3D Camera Control*. 2008. Class slides for CS-543 Course Computer Graphics. Disponível em: <http://web.cs.wpi.edu/~gogo/courses/cs543-/slides/cs543_12_Camera.pdf>. Acesso em: 28 Aug. 2012.

LENGYEL, E. Mathematics for 3d game programming and computer graphics. Charles River Media INC, Hingham Massachusets USA, 2004. Disponível em: <http://books.google.com-.br/books?id=bfcLeqRUsm8C>. Acesso em: 01 Ago 2012.

LEWIS, J. P.; PIGHIN, F.; ANJYO, K. Scattered data interpolation and approximation for computer graphics. In: *ACM SIGGRAPH ASIA 2010 Courses*. New York, NY, USA: ACM, 2010. (SA '10), p. 2:1–2:73. ISBN 978-1-4503-0527-3. Disponível em: <http://doi.acm.org/10.1145/1900520.1900522>. Acesso em: 29 Oct 2011.

MCCARTNEY, E. J. *Optics of the Atmosphere*. United States: John Wiley & Sons, 1976.

METEOPT. Www.meteopt.com. Disponível em: <http://www.meteopt.com>. Acesso em: 25 fev. 2012.

MUNSHI, A.; GINSBURG, D.; SHREINER, D. *OpenGL ES 2.0*. Boston, MA USA: Addison-Wesley, 2009. ISBN 978-0-321-50279-7.

NAMI, M. R. A comparison of object-oriented languages in software engineering. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 33, n. 4, p. 6:1–6:5, jul. 2008. ISSN 0163-5948. Disponível em: <http://doi.acm.org.ez22.periodicos.capes.gov.br/10.1145-/1384139.1384145>.

NERY, J. T. Dinâmica climática da região sul do brasil. *Revista Brasileira de Climatologia*, v. 1, n. 1, 2005. Disponível em: <http://www.geografia.fflch.usp.br/abclima/>.

NEWTON, R. G. *Scattering Theory of waves and particles*. 2nd edition. ed. East 2nd Street, Mineola, N.Y. USA: Dover, 2002.

OLIVEIRA JR, A. A.; SCHEER, S.; SATO, F. 3D visualization tool for meteorological radar data using webgl. In: *Proceedings in 10th World Congress on Computational Mechanics*. [S.l.: s.n.], 2012. ISBN 978-85-86686-70-2. DVD-ROM, Arquivo pdf WCCM2012-19264.pdf.

PRATEANO, V.; RUPP, I.; GARMATTER, B. Chuva isola o litoral do Paraná. *Gazeta do Povo*, 12 mar. 2011.

RAGGETT, D. Extending WWW to support Platform Independent Virtual Reality. 1994. Disponível em: <http://www.w3.org/People/Raggett/vrml/vrml.html>. Acesso em: 16 apr 2012.

RENDER. Wikipédia - a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki-/Renderiza%C3%A7%C3%A3o>. Acesso em: 21 apr. 2012.

RINEHART, R. E. *Radar for Meteorologists*. Nevada, MO, USA: Rinehart Publications, 2004.

ROST, R. J. et al. *OpenGL Shading Language*. 2nd. ed. Boston, MA USA: Person Education Inc, 2006. ISBN 0321334892.

RU, Y. *Volumetric Visualization Of NEXRAD Level II Doppler Weather Data From Multiple Sites*. Dissertação (Mestrado) — Purdue University, 5 dez. 2007.

SCHROEDER, W. J.; YAMROM, B. A compact cell structure for scientific visualization. In: *SIGGRAPH 94 Course Notes CD-ROM, Course 4: Advanced Techniques for Scientific Visualization*. ACM, 1994. p. 53–59. Disponível em: <www.kmh-lanl.hansonhub.com-/publications/medim94.pdf>.

SHADER. Wikipédia - a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki-/Shader>. Acesso em: 21 apr. 2012.

SILVA NETO, M. A. *Mineração Visual de Dados: Extração do Conhecimento a partir das Técnicas de Visualização da Informação e Mineração de Dados Experimentos: Itaipu e Simepar*. Dissertação (Mestrado) — Universidade Federal do Paraná, 2008.

TELEA, A. *Data visualization : principles and practice*. Wellesley, MA, USA: A.K. Peters, 2008.

TRISOTTO, F.; GERON, V.; ANIBAL, F. Chuva derruba Árvores e deixa 28 bairros no escuro. *Gazeta do Povo*, 2 abr. 2011.

WATT, A. *3D Computer Graphics*. 3rd. ed. [S.l.]: Pearson Education Limited, 2000. ISBN 0201398559.

WEB3D Consortium. X3D language bindings (ECMAScript and Java). *American Society of Civil Engineers*, abr. 2012. Disponível em: <http://www.web3d.org/x3d/specifications/x3d/>. Acesso em: 16 apr 2012.

WILSON, J. W. Precipitation nowcasting: Past, present and future. *Sixth International Symposium on Hydrological Applications of Weather Radar*, Melbourne, Australia, 2004. Disponível em: <http://www.cawcr.gov.au/bmrc/basic/old_events/hawr6/qpf-/WILSON_KEYNOTE.pdf>. Acesso em: 02 out. 2011.

APÊNDICE A – Depoimentos

Foram colhidos depoimentos para validação e testes do Visualizador.

Os depoimentos foram colhidos após a apresentação da aplicação, explicação de seu funcionamento e demonstração do uso. Para demonstração foram utilizados dois conjuntos de dados:

- Conjunto de dados da elevação de $0,5°$
- Conjunto de dados com todas as 13 varreduras com os seguintes ângulos de elevação: $0,5°$, $1,0°$, $1,5°$, $2,0°$, $3,0°$, $4.0°$, $5,0°$, $6,5°$, $8,0°$, $10,0°$, $12,0°$, $15,0°$, $18,0°$ e $21,0°$.

Para cada conjunto de dados, os dados foram visualizados e houve oportunidade de navegação com movimentos da câmera para demonstrar esta funcionalidade. Também foi utilizado o recurso de filtros dos dados, reduzindo os dados visualizados para aqueles fora da faixa de filtragem. Os valores dos filtros aplicados foram:

- -24 dBZ a 0 dBZ
- -24 dBZ a 24 dBZ
- -24 dBZ a 36 dBZ

Aos depoentes foi solicitado que respondessem uma questão objetiva : - Com base na sua experiência, a aplicação desenvolvida tem potencial de uso em ambiente operacional ?

Também foi solicitado que os depoentes fizessem comentários sobre a aplicação, mencionando recursos, limitações e sugestões.

A.1 Dr. Leonardo Calvetti

Dr. Leonardo Calvetti é Meteorologista-Pesquisador no Instituto Tecnológico Simepar. Doutor em Meteorologia pela Universidade de São Paulo (USP).

Durante a demonstração da aplicação, deu depoimento sobre a aplicação ressaltando que o Meteorologista tem sua formação baseada em aplicações e ferramentas bidimensionais. Logo, ao ser apresentado para uma ferramenta bidimensional, sempre buscam-se planos horizontais e verticais. Quando perguntado sobre se existe potencial de uso para aplicação apresentada, disse que há potencial, porém em complemento a ferramenta bidimensional, isto devido a própria formação do Meteorologista. Durante a apresentação da aplicação, Dr. Leonardo achou muito interessante a navegabilidade, com a possibilidade de aproximação e verificação dos detalhes dos dados. Diante disto, propôs ainda várias sugestões, dizendo que a interpolação é necessária para melhorar a estética da imagem formada, que dará maior clareza e compreensão, principalmente devido ao interesse em apresentar as imagens para a mídia (jornais, televisão, etc). Enfatizou que a inclusão de aspectos de relevo e de topografia, tais como bacias hidrográficas, elevações e montanhas podem melhorar na compreensão de fenômenos severos (tempestades e chuvas intensas), pois pode-se verificar se a formação do evento ocorreu por influência das circulações "vale-montanha" dos ventos. Ainda, é possível perceber com mais clareza fenômenos típicos de regiões, tais como formação de chuvas orográficas [1] do região da Serra do Mar e do litoral.

Dr. Leornardo sugeriu ainda a inclusão de cidades e alteração de pontos de monitoramento e observação e implementar ao sistema capacidade para estimar altitudes para que se possa determinar diferença de altitudes entre a base da nuvem e o topo da nuvem, que são dados interessantes para compreensão de fenômenos. Outras sugestões incluem, a exportação de animação da navegação e de possibilidade de tirar uma foto da visualização corrente. Estas sugestões estão ligadas ao potencial de uso da aplicação para o estudo dos fenômenos, principalmente dos severos.

Dr. Leonardo Calvetti

[1] Chuva orográfica ou "chuva de relevo" é chuva causada pela mudança de altitude de camadas de ar devido a montanhas ou encostas, que forçam estas camadas a subir, provocando resfriamento da massa de ar e acarretando a chuva. São típicas de regiões serranas ou terrenos com encostas.

A.2 Dr. Reinaldo Silveira

Dr. Reinaldo Silveira é Coordenador de Integração Tecnológica no Instituto Tecnológico Simepar. Doutor em Matemática Aplicada pela University of Essex, Inglaterra.

Durante a demonstração da aplicação achou a aplicação interessante. Comentou que a aplicação tem potencial de utilização tanto em pesquisa quanto no ambiente operacional. Segundo ele, a visualização tridimensional dos dados puros permitiu perceber os próprios artefatos do radar (Radar Artifacts)[2] que muitas vezes não são percebidos em uma visualização bidimensional. Mencionou que um fator limitante é a velocidade de navegação da câmera quando da apresentação de todos os bins de todas as elevações (4.032.000 bins), mas sugeriu que vale o estudo de melhorias deste item, bem como alteração da posição da câmera e melhorar a aplicação da ferramenta de filtro, além da ferramenta já disponível.

Dr. Reinaldo Bomfim da Silveira

[2]Radar Artifacts ou artefatos do radar são anomalias que acontecem na reflexão das ondas devido a variação do meio de propagação. São dificeis de interpretar. Um dos exemplos básicos é que se um eco de grande intensidade capturado, ecos "após" este eco podem ser dificeis de capturar, pois o conjunto de hidrometeoros que causou a alta refletividade, acabam por atrapalhar a coleta dos dados após este conjunto.

A.3 Fábio Sato

Fábio Sato é Coordenador do Núcleo de Informática no Instituto Tecnológico Simepar. Engenheiro Civil com Pós Graduação em Tecnologia de Sistemas de Informação. Sr. Fábio Sato comentou que a tecnologia WebGL já pode ser considerada para desenvolvimento de aplicações de Visualização Científica. Quanto ao protótipo, mencionou:

> O protótipo desenvolvido permitiu avaliar desempenho e usabilidade de uma aplicação totalmente Web para radar meteorológico. A abordagem de visualização tridimensional de dados através da representação via Grid Estruturado Esférico nos deu algumas idéias de como esta técnica de visualização poderá ser utilizada em conjunto com as técnicas tradicionais em duas dimensões. Temos idéia de dar continuidade ao trabalho, explorando melhor as questões de desempenho e transferência de dados.

_____

Fábio Sato

APÊNDICE B – JSON

JSON - Javascript Object Notation ou Notação de Objetos Javascript é um formato para troca de dados. É fácil de ser lido ou escrito tanto por computadores quanto por seres humanos ((INTRODUCING. . . , 2012), (CROCKFORD; IETF, 2006)). É um sub conjunto de JavaScript ECMA 262 3rd edition de 1999 (ECMA, 2011). Apesar deste vínculo com JavaScript, é desvinculado desta linguagem, sendo utilizado em C, C++, Java, Python entre outras.

É baseado em 2 estruturas principais :

- Coleção de pares de nomes e valores
- Lista de valores ordenados, sendo identificado como matrizes ou vetores ou sequência

Objetos são definidos pelo conteúdo entre e . Os membros de um objeto são definidos entre as chaves, podendo conter outros objetos.

Membros são pares de nome e valor, separados por :. Vários membros de um objeto são separados por vírgula. O nome define o nome do atributo ou propriedade do objeto, que é um string, enquanto que o valor é o valor desta propriedade. O nome é um string, que é definida entre “ e ”. Os valores podem ser:

- Strings
- Números
- Objeto
- valor booleano true ou false
- null
- matriz

Matrizes são estruturas que contém valores apenas contidos entre colchetes [] e separados por vírgula.

Exemplos: Objeto contendo um atributo String “nome”:“Simples teste”

Objeto contendo atributo numérico: “salario”:1000.00

Objeto contendo atributo matriz (ou array): “dados”:[10,20,30]

Objeto contendo uma coleção (matriz) de outros objetos: “empregados”:[“nome”:“Fulano de tal”,“nome”:“Siclano de tal ”, “nome”:“José da Silva”]

Os objetos definidos devem ser atribuídos a variáveis para que possam ser utilizados. E ao objeto que é atribuído, os pares de valores são acessados diretamente pelo nome. Exemplo: var t1 = “nome”:“Outro simples teste”

No código JavaScript, acessando t1.nome, retornará o valor contido no atributo nome, no caso, *Outro simples teste*.

APÊNDICE C – Código-fonte Visualizador

O código-fonte está disponível em formato eletrônico disponível através de cd-rom depositado na Biblioteca Central da UFPR - Universidade Federal do Paraná.